Representation Independent Proximity and Similarity Search

Yodsawalai Chodpathumwan *ychodpa@illinois.edu*[1], Arash Termehchy *termehca@oregonstate.edu*[2],
Amirhossein Aleyasen *aleyase2@illinois.edu*[1] and Yizhou Sun *yzsun@ccs.neu.edu*[3]

[1] *Department of Computer Science, University of Illinois, Urbana, IL, USA*
[2] *School of EECS, Oregon State University, Corvallis, OR, USA*
[3] *College of Computer and Information Science, Northeastern University, Boston, MA, USA*

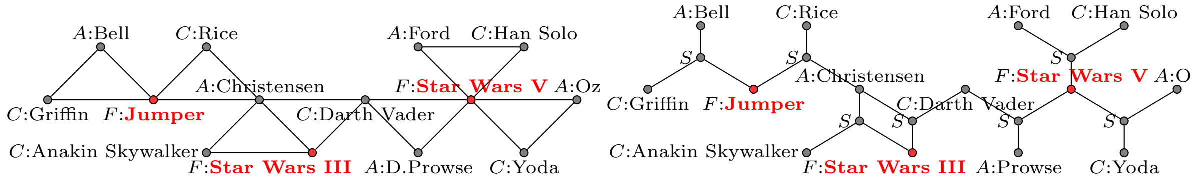
(Dated March 1, 2016)

Abstract Finding similar entities is a fundamental problem in graph database management and analytics. Similarity search algorithms usually leverage the structural properties of the database to quantify the degree of similarity between entities. However, the same information can be represented in many different structures and the structural properties observed over particular representations may not hold for alternative structures. Thus, these algorithms are effective on some representations and ineffective on others. We define the property of representation independence for similarity search algorithms as their robustness against transformations that modify the structure of databases and preserve their information content. We introduce two widespread groups of such transformations called *relationship reorganizing* and *entity rearranging*. We propose an algorithm called **R-PathSim**, which is provably robust under relationship-reorganizing and a subset of entity-rearranging transformations. Our empirical results show that the output of current algorithms except for R-PathSim are highly sensitive to the data representation and R-PathSim is as efficient as and as effective or more effective than other algorithms.

1 Introduction

Finding similar or strongly related entities is a fundamental and important problem in graph data management and analytics [15, 25, 35, 29, 18, 24, 8, 19, 31, 23, 2, 32]. It is a building block of algorithms for various important database management and analytics problems, such as similarity query processing [15, 23, 2], pattern query matching [30, 27, 17], community detection [15, 28]. and link prediction [20]. Since the properties of *similar* or *related* entities cannot be precisely defined, current similarity and proximity search algorithms use intuitively appealing heuristics that leverage information about the links between entities. For instance, *Random Walk with Restart* (RWR) quantifies the degree of similarity or relevance between two entities as the likelihood that a random surfer visits one of the entities in the database given it starts and keeps re-starting from the other entity [29]. *SimRank* evaluates the similarity between two entities according to how likely two random surfers will meet each other if they start from the two entities [15]. Figure 1a shows fragments of IMDb (*imdb.com*), which contains information about movies, actors, and characters. To represent the relationship between a character, its movie, and the actor who played the character IMDb connects these entities through some edges. Assume that a user asks for the most similar movie to *Star Wars III* in Figure 1a. Since the RWR and SimRank score of *Star Wars V* (RWR-score = 0.061, SimRank-score = 0.213) are larger than those of *Jumper* (RWR-score = 0.060, SimRank-score = 0.185), RWR and SimRank find *Star Wars III* more similar to *Star Wars V* than to *Jumper*, which is arguably an effective answer.

The power of similarity search algorithms, however, remains out of the reach of most users as today's similarity search algorithms are usable only by trained data analysts who can predict which algorithms are likely to be effective for particular databases. To see why, consider the excerpts of Freebase (*freebase.com*) in



(a) IMDb (b) Freebase

Figure 1: Fragments of IMDb and Freebase, where A, C, F, and S refer to *actor*, *character*, *film* and *starring*, respectively.

Figure 1b. Figure 1b contains information about exactly the same set of entities and relationships as Figure 1a. It differs with Figure 1a only in how it represents the relationships between a character, its movie, and its actor: it connects them to a common node labeled *starring*. Hence, it contains essentially the same information as Figure 1a. Database researchers have recognized that different, i.e., non-isomorphic, structures can contain the same information [1, 7]. As opposed to their results over Figure 1a, RWR and SimRank find *Star Wars III* more similar to *Jumper* (RWR-score = 0.014, SimRank-score = 0.076) than to *Star Wars V* (RWR-score = 0.011, SimRank-Score = 0.074) in Figure 1b.

Generally, there is no canonical representation for a particular set of content and people often represent the same information in different structures [1]. Thus, users have to restructure their databases to some *proper representation(s)*, to effectively use similarity and proximity search algorithms, i.e., deliver the answers that a domain expert would judge as relevant. To make matters worse, these algorithms do not normally offer any clear description of their desired representations and users have to rely on their own expertise and/or do trial and error to find such representations. Further, the structure of large-scale databases constantly evolve and we want to move away from the need for constant expert attention to keep our algorithms effective.

One approach to solve the problem is to run a similarity search algorithm over all possible representations of a data set and select the representation(s) with the most accurate answers. Nevertheless, because most similarity algorithms are unsupervised, there is no validating data available to measure the effectiveness of these algorithms over various representations. Moreover, it is generally undecidable to compute all possible representations of a database [7]. If we restrict the set of possible representations, a database may still have enormous representational variations. For example, the number of vertical decompositions of a relational table may be exponential in terms of the number its attributes [1]. Further, as graph databases have less restrictive schemas than relational databases, they may have more representational variations and need more time to generate and run algorithms over them. Researchers have proposed the idea of *universal relation* to achieve some level of schema independence for SQL queries over relational databases [1]. One may extend this idea and define a *universal representation* in which all graph databases can be represented and develop similarity search algorithms that are effective over this representation. Nevertheless, the experience gained from the idea of universal relation, indicates such representation may not always exist [1]. Further, it may not be practical to force developers to represent their data in and create their algorithms for a particular format.

In this paper, we propose the property of *representation independence* for similarity search algorithms over graph data, i.e., the ability to deliver the same answers regardless of the choices of structure for organizing the data. To the best of our knowledge, the property of representation independence has not previously been explored for similarity search algorithms and/or graph databases. We believe that the key to the success of building representation independent analytics in general and similarity search algorithms in particular is to modify current algorithms to become representation independent instead of developing new representation independent algorithms from the scratch. Current well-known similarity search algorithms have already been adapted in both academia and industry to solve various graph analytics problems. Hence, it is easier for organizations to modify these algorithms rather than using new algorithms. They have been shown empirically to be effective over some databases, which provide evidences that their reasonable modifications may be effective over even more databases. Our contributions are as follows.

- We introduce and formally define the representation independence of a similarity search algorithm as its robustness under transformations that modify the structure of its input database but preserve its information content.
- We introduce a widespread group of transformations called *relationship-reorganizing transformations* that modify the representation of relationships between entities in a database. We show that current similarity search algorithms are not representation independent under relationship-reorganizing transformations. We extend a current similarity search algorithm called PathSim and develop a new algorithm called *Robust-PathSim* (*R-PathSim* for short). We prove that R-PathSim is representation independent under relationship reorganizing-transformations.
- We introduce another group of common transformations that reposition entities in a database called *entity-rearranging transformations*. We show that current similarity search algorithms including R-PathSim are not representation independent under this family of transformations. We extend R-PathSim and prove that its extension is robust under entity-rearranging transformations.
- We empirically study the representation independence of well-known similarity search algorithms under relationship reorganizing and entity-rearranging transformations using several real-world databases and transformations. Our results indicate that relationship-reorganizing and entity-rearranging transformations considerably affect the results of all algorithms but R-PathSim. We also empirically evaluate the effectiveness and efficiency of R-PathSim using real-world databases and show that it is as effective or more effective than

and as efficient as current similarity search algorithms.

This paper is organized as follows. Section 2 provides the background and Section 3 defines the property of representation independence. Section 4 explores relationship-reorganizing transformations and describes R-PathSim. Section 5 introduces entity-rearranging transformations and extends R-PathSim. Section 6 contains our empirical results.

2 Background

2.1 Related Work

The architects of relational models envisioned the desirable property of *logical data independence*. Oversimplifying a bit, this meant that an exact query should return the same answers regardless of the schema chosen for the data [1, 6]. One may consider the idea of representation independence as an extension of the principle of logical data independence for similarity and proximity search algorithms. Nevertheless, these ideas differ in an important aspect. One may achieve logical data independence for database applications by creating a set of views over the database, which keep the application unaffected from modifications in the database schema [1]. However, characteristics of the ideal representations for similarity and proximity search algorithms are not clearly defined. Also, graph databases follow far less rigid schemas and are more amenable to change than relational databases. Hence, it takes far more time and more in-depth expertise to find the proper representation as well as create and maintain the mapping between the database and this representation.

Researchers have proposed keyword query interfaces over tree-shaped XML data that return the same answers to a keyword query over databases with equivalent content but different choices of structure [26]. We, however, introduce and study the concept of representation independence for a different problem and data model. The task of similarity and proximity search has a different semantic than keyword search and requires different types of algorithms. Further, graph databases are more complex than tree-shaped XML databases and offer novel challenges in defining the concept of representation independence and developing representation independent algorithms.

Researchers have also analyzed the stability of random walk algorithms in graphs against relatively small perturbations in the data [22, 10, 5]. We also seek to instill robustness in graph mining algorithms, but we are targeting robustness in a new dimension: robustness in the face of variations in the representation of data. Researchers have provided systems that help users with transforming and wrangling their data [16, 13, 9, 33]. We also address the problem of data preparation but using a difference approach: *eliminating the need to wrangle the data*.

Researchers have proposed several normal forms for relational and tree-shaped XML schemas [1, 3, 34]. Nevertheless, we focus on finding representation independent similarity search algorithms rather than transforming the database to a particular representation with some desirable properties. Moreover, because similarity search algorithms usually operate over graph databases without rigid schemas, our transformations are defined over a much less restrictive schemas than relational schemas or XML DTDs. Our entity-rearranging transformations somewhat resemble normalization/ denormalization in relational and tree-shaped XML databases. Our transformations, however, modify the connections between entities in the database instead of creating or removing duplicates. They are also defined over graph databases rather than relational or tree-shaped databases.

Blank nodes represent the existence of resources without any global identifier, i.e., existential variables, in RDF databases [11, 14]. As blank nodes often convey redundant information, researchers have proposed methods to remove them from RDF databases [11, 14]. However, our goal in this paper is not to remove certain nodes from a database. Further, because our databases do not contain any existential variable, we use a different approach to ensure that our transformations do not modify the information content of a database. For instance, as opposed to our transformations, the mappings that eliminate blank nodes may not be invertible. Some serializations of RDF data, such as RDF/XML, may assign labels and identifiers within the scope of a document to blank nodes in the document [14]. Our framework covers these applications of blank nodes. Nevertheless, it also addresses the representational shifts over databases that do not contain any blank node. Researchers have proposed algorithms to convert RDF data sets that contain certain relationships in *RDF Schema*, such as *rdfs:subClassOf*, to some normal forms [11]. Our transformations, however, are not limited to particular set of relationships.

Schema mapping has been an active research area for the last three decades [1]. In particular, researchers have defined schema mappings over graph databases as constraints in some graph query language in the context of data exchange [4]. As opposed to the transformations in our work, the original and transformed databases in those settings may not represent the same information. We also focus on evaluating the representation independence of similarity search algorithms rather than traditional questions in schema mapping and data exchange, such as computing the transformed database instances.

2.2 Data Model

Let *dom* be a fixed and countably infinite set of values. To simplify our definitions, we assume the members of *dom* are strings. Let L be a finite set of labels. Each member of L denotes a *semantic type* in a domain of interest, e.g. *actor* and *film* in movie domain. A database D defined over L is a graph $D = (V, E, \mathcal{L}, \mathcal{A})$, where V is the set of nodes, $E \subseteq V \times V$ is the set of edges, $\mathcal{L}$ is a total function from V to L that assigns a label to each node, and $\mathcal{A}$ is a function from V to *dom* that assigns values to nodes in V. We denote the set of all databases whose labels belong to L as $\mathbf{L}$.

Real-world graph databases often contain nodes without any value to represent relationships between or categorize entities [12, 14]. Figure 1b is an example of using nodes without values to represent relationships between entities. One may use these types of nodes for several reasons. It is sometimes easier to express relationships between relationships in a database using nodes without values, e.g., *starring*, [12]. For example, consider a database that contains information about various types of artists, such as painters. The relationship *paints* between a painter and her paintings is a subclass of the relationship *creates* between an artist and its creations. To capture the *subclass* relationship between relationships *paint* and *create*, one may represent *creates* and *paints* as nodes without any value and connect them by an edge or through another node that represents the relationship *subclass-of.* Also, one may use nodes without values to categorize related nodes or express complex relationships which help users understand the structure of the database more easily. For example, RDF data sets often use nodes without any value and global identifier, i.e., *blank nodes*, to represent complex relationships between entities [14]. Empirical studies using 1.23 billion RDF triples and 8.37 million RDF documents collected from the Web indicate that 30% of RDF triples and 44.9% of RDF documents contain blank nodes [14]. Following the terminology used in similarity search literature, we call the nodes with values *entities* [15, 25]. We assume that each set of labels L has two mutually exclusive and collectively exhaustive subsets of N, which contain labels for entities, and R, which contain labels for nodes without values. That is, nodes whose labels are in R do not have any value in databases of $\mathbf{L}$.

We denote a similarity query q, query for short, over database D as (v), where v is an entity node in D. Query $q = (v)$ seeks for entity nodes other than v in D that are similar to v [25, 15, 29, 28]. For example, query ($film$:Star Wars III) over the database fragment shown in Figure 1b asks for other entities similar or strongly related to the node $film$:Star Wars III in the database. Given query q over database D, a similarity search algorithm returns a ranked list of entity nodes from D, i.e., the result of q. For example, the result of query ($film$:Star Wars III) over Figure 1b could be the list of entities $film$:Star Wars V and $film$:Jumper. We denote the result of query q over database D using similarity search algorithm S by $q_S(D)$. If S is clear from the context, we denote $q_S(D)$ as $q(D)$.

3 Representation Independence

A representation-independent similarity search algorithm should return the same list of entities for the same query across databases that represent the same information. It is important to precisely define the conditions under which two graph databases represent the same information. Researchers have defined the conditions under which relational or XML schemas represent the same information [7, 1, 26]. Graph databases, however, do not generally follow strict schemas. Hence, we extend the ideas on comparing information contents of databases for our data model.

Transformation T is a function from a set of databases $\mathbf{L}$ to a set of databases $\mathbf{K}$, denoted as $T : \mathbf{L} \rightarrow \mathbf{K}$. For instance, consider set of labels $L_1 = \{actor, film, char\}$ and $L_2 = \{actor, film, char, starring\}$. The databases in Figures 1a and 1b belong to $\mathbf{L_1}$ and $\mathbf{L_2}$, respectively. One may define transformation $T_{IMDb2Freebase} : \mathbf{L_1} \rightarrow \mathbf{L_2}$, which replaces every triangle between the nodes of labels *film*, *character*, and *actor* with a subgraph whose nodes have the same labels and values of the nodes in the triangle and are connected to a single new node with the label *starring.* This transformation maps the database in Figure 1a to the database in Figure 1b.

A transformation T is invertible if a database D is reconstructible from information in database $T(D)$. For example, transformation $T_{IMDb2Freebase}$ is invertible as the original database in Figure 1a can be reconstructed using the information in its transformed one, e.g., Figure 1b. However, a transformation that removes the edges between each *film* node and its neighboring *actor* and *character* nodes from Figure 1a is not invertible because there is insufficient information in the transformed database to recover the relationship between *film*, *actor*, and *character* nodes. More formally, a data graph $D = (V, E, \mathcal{L}, \mathcal{A})$ and $D' = (V', E', \mathcal{L}', \mathcal{A}')$ are **isomorphic** ($D \cong D'$) iff there is a bijection $f : V \rightarrow V'$ such that (1) $\forall v \in V$, $\mathcal{L}(v) = \mathcal{L}'(f(v))$ and $\mathcal{A}(v) = \mathcal{A}'(f(v))$, and (2) $\forall u, v \in V$, $(u, v) \in E$ iff $(f(u), f(v)) \in E'$. Isomorphic databases contain exactly the same set of nodes and connectivity between nodes. In the followings, given isomorphic databases D and D', we say that D and D' are the same database. A transformation $T : \mathbf{L} \rightarrow \mathbf{K}$ is **invertible** iff there is a transformation $T^{-1} : \mathbf{K} \rightarrow \mathbf{L}$ such that, for all $D \in \mathbf{L}$, we have $T^{-1}(T(D)) \cong D$. Since the transformed database of an invertible transformation contains sufficient information to build the original database, the original and transformed databases contain essentially the same information [1, 7]. As depicted in Figure 1, the original and transformed databases of an invertible transformation are **not** generally isomorphic.

To precisely define representation independence over a transformation T, we should make sure that users can pose the same set of queries over databases D and $T(D)$. Similarity search queries over a database D are entities of D, thus, D and $T(D)$ should essentially contain the same set of entities. Moreover, similarity search algorithms generally view the labels of nodes as their semantic types [25]. For example, they assume that the nodes with label *film* in Figure 1a represent entities from the same semantic type, while the nodes of label *film* and *actor* belong to different semantic types. They use these pieces of information to find similar nodes more accurately. Thus, for these algorithms to return the same results over a transformation T, T should map entities of the same label in the original database D to entities with the same label in the transformed database $T(D)$. We consider two data values equal iff they are lexicographically equal: they have the same length and contain the same characters in the same positions. Our approach can also support other definitions of equality between data values.

Definition 3.1. *Transformation $T : \mathbf{L} \rightarrow \mathbf{K}$ that transforms database $D = (V, E, \mathcal{L}, \mathcal{A})$ to $T(D) = (V_T, E_T, \mathcal{K}, \mathcal{A}_T)$ is entity preserving iff there is a bijective mapping M between entities in V and V_T such that*

- *For all entities $v \in V$, we have $A(v) = A_T(M(v))$.*
- *For all entities $v_1, v_2 \in V$ that $\mathcal{L}(v_1) = \mathcal{L}(v_2)$, we have $\mathcal{K}(M(v_1)) = \mathcal{K}(M(v_2))$.*

For example, transformation $T_{IMDb2Freebase}$ is entity preserving as it does not introduce any new entity to or remove any entity from its input databases. An entity preserving transformation T provides a bijective mapping between every entity over D to an entity over $T(D)$. By the abuse of notation, we denote the entity in database $T(D)$ that is mapped to the entity v in database D, as $T(v)$. To simplify our definitions and proofs, we assume that transformations do not rename the labels in databases. Our results extend for the transformations that rename labels.

If a transformation is both invertible and entity preserving, it is *similarity preserving*. Each similarity preserving transformation T maps a databases D to a database $T(D)$ that has the same information and the same set of possible queries as D. It further guarantees that the entities of the same semantic type in D share the same label in $T(D)$. Hence, it is possible to design an effective similarity search algorithm that returns essentially the same answers for every query over D and $T(D)$. Because answers of similarity search algorithms are normally in the form of ranked list of entities, we define a representation independent similarity search algorithm as follows.

Definition 3.2. *Similarity search algorithm S is representation independent under similarity preserving transformation $T : \mathbf{L} \rightarrow \mathbf{K}$ iff for each database $D \in \mathbf{L}$ and $T(D) \in \mathbf{K}$ and every query q over D, there is a bijective mapping N between $q(D)$ and $T(q)(T(D))$ such that*

- *for all entities $v \in q(D)$ and $N(v) \in T(q)(T(D))$, we have $N(v) = T(v)$*
- *entity v appears before entity u in $q(D)$ iff $N(v)$ ranks before $N(u)$ in $T(q)(T(D))$.*

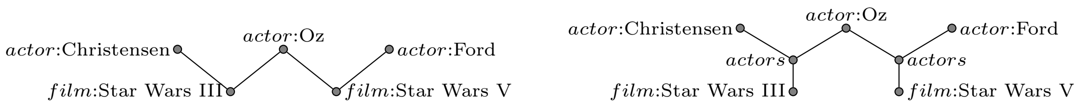
The first condition in Definition 3.2 guarantees that the answers to query q over databases D and $T(q)$ over $T(D)$ contain the same set of entities. Its second condition ensures that these entities appear at the same order in results of q and $T(q)$ over D and $T(D)$, respectively. According to Definition 3.2, if answers v and u tie, i.e., are placed at the same position, in $q(D)$, $T(v)$ and $T(u)$ must also tie in $T(q)(T(D))$.

The result of a query is a list of entities, where each entity is shown by its semantic type and value. A database may have several entities with equal values from of the same semantic type. Hence, it may not be possible to check the first condition of Definition 3.2 using only the semantic types and values of the entities in the results of a query. One may assign a unique (printable) id to each entity in the database to address this problem [7]. To simplify our framework and definitions, we assume that databases do not contain entities that belong to the same semantic type and have equal values. Our results extend for other cases.

4 Relationship Reorganization

4.1 Relationship-Reorganizing Transformations

Generally speaking, a relationship-reorganizing transformation T maps database D to database $T(D)$ such that D and $T(D)$ contain the same set of entities and relationships, but they may represent these relationships in different forms. More specifically, D and $T(D)$ may express the same relationship between the same set of entities using some edges or some nodes without values. For example, Figure 2b uses a set of edges to represent the relationship between a movie and its actors. However, Figure 2a expresses the same relationship between



(a) IMDb (b) Movielicious (www.netwalkapps.com/movies-xml-format)

Figure 2: Fragments of movie databases.

the same set of entities by a node without value, i.e., *actors*. In this section, we formally define this type of representational variation. First, we find patterns that represent relationships between entities in a database. Then, we define the conditions under which two patterns represent the same information. Finally, we define a relationship-reorganizing transformation as a bijective mapping between patterns that represent the same information in the original and transformed databases.

A **walk** in database is a sequence of nodes and edges where each edge's endpoints are the preceding and following nodes in the sequence. We show a walk in database D as a sequence of nodes $[v_0, \ldots, v_n]$, such that v_i are nodes and (v_{i-1}, v_i) ,$0 \leq i \leq n$, are edges in D. For example, w_1 = [*actor*:Ford, *actors*, *film*:Star Wars V] is a walk in Figure 2a. Intuitively, a walk represents some relationship between its entities. For example, walk w_1 in Figure 2a shows that actor *Ford* has played in movie *Star Wars V*. One may use paths to capture relationships between entities in a database [25]. But, we show in Section 5 that walks represent more varieties of relationships than paths, which enables us to achieve representation independence over more transformations. To simplify our framework, we assume that each database is a simple graph: it has at most one edge between each two nodes and does not have any loop at each node. Our framework extends for other cases. We are interested in walks that express relationships between entities. Hence, we consider only walks that start and end with entities.

Some walks contain consecutive forward and backward traverses from an entity to a node without value. For example, walk [*actor*:Ford, *actors*, *film*:Star Wars V, *actors*, *film*:Star Wars V] in Figure 2b expresses the relationship between actor *Ford* and movie *Star Wars V*. It contains consecutive forward and backward traverses from entity *film*:Star Wars V to the node without value *actors*. The information expressed by this walk can be represented using a shorter walk [*actor*:Ford, *actors*, *film*:Star Wars V], which does not contain any consecutive forward and backward traverses from *film*:Star Wars V to *actors*. Another example of such walks in Figure 2b is [*film*:Star Wars V, *actors*, *film*:Star Wars V]. This walk does not provide any information regarding the relationships between entities in the database. Hence, unless otherwise noted, we consider only walks that does *not* have any consecutive forward and backward traverses from an entity to a node without value because they do not contain any information regarding the relationship between entities or their information can be expressed by shorter walks.

The **meta-walk** of a walk $[v_1, \cdots, v_n]$ in database $D = (V, E, \mathcal{L}, \mathcal{A})$ is a sequence of labels $[\mathcal{L}(v_1), \cdots, \mathcal{L}(v_n)]$. For example, the meta-walk of walk [*actor*:Ford, *actors*, *film*:Star Wars V] in Figure 2b is [*actor*, *actors*, *film*]. Each meta-walk represents a pattern of relationship between entities of certain semantic types. Some meta-walks represent basically the same relationships between the same sets of semantic types. For instance, meta-walk [*actor*, *film*] in Figure 2a and meta-walk [*actor*, *actors*, *film*] in Figure 2b represent the relationship of starring in movies between the same set of actors and movies. Next, we define the conditions under which two meta-walks represent the same relationship between the same set of entities. Given database $D = (V, E, \mathcal{L}, \mathcal{A})$, the value of an entity node $e \in V$ is the pair $\mathcal{L}(v) : \mathcal{A}(v)$. The value of a walk $w = [v_0, \ldots, v_n]$ is the tuple $[a_0, \ldots, a_m]$, $m \leq n$ such that a_0 and a_m are the values of v_0 and v_n, respectively, and for all $0 \leq i < j \leq n$ and $0 \leq i', j' \leq m$ if $a_{i'}$ and $a_{j'}$ are the values of entity nodes v_i and v_j, respectively, then $i' < j'$. For instance, the value of walk [*actor:*Ford, *actors*, *film*:Star Wars V] is [*actor:*Ford, *film*:Star Wars V]. Values of two walks are **equal** iff they have equal arities and their corresponding positions contain the same label and equal values. Two walks are **content equivalent** iff their values are equal. For instance, walk [*actor*:Ford, *film*:Star Wars V] in Figure 2a and walk [*actor*:Ford, *actors*, *film*:Star Wars V] in Figure 2b are content equivalent. We show content-equivalent walks w and x as $w \equiv x$. Let $p(D)$ denote the set of walks in database D whose meta-walk is p.

Definition 4.1. *Meta-walks p_1 in database D_1 and p_2 in database D_2 are* **content equivalent** *iff there is a bijection $M : p_1(D_1) \rightarrow p_2(D_2)$ where for all $w \in p_1(D_1)$, $w \equiv M(w)$.*
Meta-walks [*actor*, *film*] in Figure 2b and [*actor*, *actors*, *film*] in Figure 2a are content equivalent. We denote content-equivalent meta-walks p_1 and p_2 as $p_1 \equiv p_2$.

Naturally, content-equivalent meta-walks represent the same sets of relationships between the same sets of entities. Thus, if a transformation bijectively maps each meta-walk in database D_1 to its content-equivalent meta-walk in database D_2, D_1 and D_2 represent the same information. We formally prove this intuition later in the section. However, this straightforward definition ignores some interesting transformations. For example, intuitively the databases in Figure 2a and Figure 2b contain the same information. But, there is not any meta-walk in Figure 2a that is content equivalent to meta-walk p_3=[*actor*, *actors*, *actor*] in Figure 2b. By looking closely at the Figure 2b and original Movielicious data, we observe that the node *actors* always groups actors that play in the same movie. Thus, each walk of p_3 is a part of a walk of meta-walk p_4= [*actor*, *actors*, *film*, *actors*, *actor*] in Figure 2b. Hence, if a transformation maps p_4 to a content-equivalent meta-walk in Figure 2a, it also preserves the information of p_3. Generally, some meta-walks contain other meta-walks. If a transformation preserves the information of a meta-walk, it will preserve the information of its contained meta-walks. Let us formalize this relationship between meta-walks. A walk w is a **subwalk** of walk x, shown as $w \sqsubseteq x$, iff w is a subsequence of x. For example, walk $[v_1, v_2, v_3]$ is a subwalk of walk $x_1 = [v_1, v_2, v_4, v_2, v_3]$. But, walk $[v_1, v_3]$ is not a subwalk of x_1 because the edge (v_1, v_3) is not in x_1. Meta-walk p is a subwalk of meta-walk r, denoted as $p \sqsubseteq r$, iff a walk of p is a subwalk of a walk of r. For example, [*actor*, *actors*, *actor*] is a subwalk of [*actor*, *actors*, *film*, *actors*, *actor*] in Figure 2b.

Definition 4.2. *Given meta-walks p and p' in database D, p'* **includes** *p iff*

- *there is a bijection M between $p(D)$ and $p'(D)$ such that for every walk $w \in p(D)$, we have $w \sqsubseteq M(w)$ and w and $M(w)$ start at the same node and end at the same node.*
- *there exists an entity label l whose occurrence in p' is more than in p, and the closest entity labels to the left and to the right of l in p' are not l.*

For example, meta-walk [*actor*, *actors*, *film*, *actors*, *actor*] includes [*actor*, *actors*, *actor*] in Figure 2b. A meta-walk p in database D is **maximal** iff it has a walk in D and it is not included in any other meta-walk. For instance, [*actor*, *actors*, *film*, *actors*, *actor*] is maximal in Figure 2a. Maximal meta-walks subsume the information of non-maximal meta-walks. Thus, if a transformation preserves only the information of maximal meta-walks in a database, it will preserve the information content of the database. Let $\mathcal{P}(\mathbf{L})$ denote the set of all meta-walks in the set of databases $\mathbf{L}$. Similarly, we denote the set of all maximal meta-walks in $\mathbf{L}$ as $\mathcal{P}_{\max}(\mathbf{L})$.

Definition 4.3. *Transformation $T : \boldsymbol{L} \rightarrow \boldsymbol{K}$ is* **relationship reorganizing** *iff there is a bijective mapping $M : \mathcal{P}_{\max}(\boldsymbol{L}) \rightarrow \mathcal{P}_{\max}(\boldsymbol{K})$ such that $p \equiv M(p)$.*

The transformations that map Figure 2b to Figure 2a and Figure 1a to Figure 1b are relationship-reorganizing.

Theorem 4.4. *Every relationship-reorganizing transformation is similarity preserving.*

Proof. Let $T : \mathbf{L} \rightarrow \mathbf{K}$ be a relationship-preserving transformation and $M : \mathcal{P}_{\max}(\mathbf{L}) \rightarrow \mathcal{P}_{\max}(\mathbf{K})$ be the bijection that T establishes between maximal meta-walks in $\mathbf{L}$ and $\mathbf{K}$. Let M^{-1} be an inverse of M. Let us define T' to be a transformation from $\mathbf{K}$ to $\mathbf{L}$ as follows. Because M is bijective, for each $D' \in \mathbf{K}$, define T' to bijectively map a maximal meta-walk p' in D' to a maximal meta-walk $M^{-1}(p')$ in $T'(D')$ s.t. $p' \equiv M^{-1}(p')$. Since we assume in Section 3 that we do not distinguish between isomorphic databases, we show that, $\forall D \in \mathbf{L}$, $T'(T(D))$ and D are the same.

For each maximal meta-walk p in D, there exists exactly one maximal meta-walk $M(p)$ in $T(D)$ s.t. $p \equiv M(p)$. For each maximal meta-walk $M(p)$ in $T(D)$, there exists exactly one maximal meta-walk $M^{-1}(M(p))$ in $T'(T(D))$ s.t. $M^{-1}(M(p)) \equiv M(p)$. Thus, $p \equiv M^{-1}(M(p))$. For each $w \in p(D)$, there exists exactly one walk $w' \in M^{-1}(M(p))(T'(T(D))$ s.t. $w \equiv w'$. Because M^{-1} is an inverse of M, $M^{-1}(M(p)) = p$, and so $w' = w$. Hence, there exists a bijection that maps a walk of some maximal meta-walk in $T'(T(D))$ to the same walk in D. Therefore, the sets of walks of maximal meta-walks in D and $T'(T(D))$ are the same.

We show that the set of all nodes that appear in a walk of some maximal meta-walk is the set of all nodes in the database. Consider that a node v in a database must appear in a walk of some meta-walk p. If p is not maximal, then p is included in some maximal meta-walk p'. That is, v appears in a walk of some maximal meta-walk in the database. Thus, the set of all nodes that appear in a walk of some maximal meta-walk is the same as the set of all nodes in the database. Using similar arguments, we prove that the set of all edges that appear in a walk of some maximal meta-walk in a database is the set of all edges in the database. Since the sets of walks of maximal meta-walks in D and $T'(T(D))$ are the same, the sets of nodes and edges in D and $T'(T(D))$ are also the same. Hence, D and $T'(T(D))$ are the same. Therefore, T is invertible.

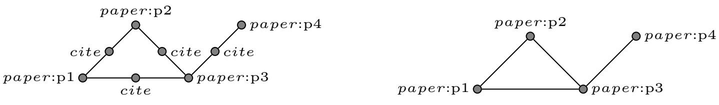
For each entity e in a database $D \in \mathbf{L}$, assume e appears in a walk w of some maximal meta-walk in D. Using Definition 4.3, T bijectively maps w to a walk w' of some maximal meta-walk in $T(D)$ s.t. $w \equiv w'$. Thus, there must exist an entity in $T(D)$ with the same label and value as e. Similarly, if an entity f in $T(D)$ exists, then f also exists in D. Hence, T is entity preserving. Therefore, T is similarity preserving. □

4.2 Toward Robust Similarity Algorithms

To the best of our knowledge, the most frequently used methods for similarity search on graph database are based on random walk, e.g., RWR [29], pairwise random walk, e.g., SimRank [15] and P-Rank [35], or relationship-constrained framework, e.g., PathSim [25, 24]. There are other similarity measures, such as common neighbors, $Katz_\beta$ measure, hitting time, and commute time, which can be considered as special cases of aforementioned heuristics. Hence, we discuss similarity search methods based on these three frameworks.

Methods that use random walk and pairwise random walks leverage the topology of a graph database to measure the degree of similarities between entities. A relationship-reorganizing transformation may remove many edges from and add many new nodes and edges to a database. Thus, it may radically modify the database topology. For example, a relationship-reorganizing transformations may drastically change the degree of a node and modify the probability that random surfers visit the node. Hence, these methods cannot always return the same answers over the original and the transformed database for the same query. In Section 1, we have shown that RWR and SimRank return different results over a database and its relationship reorganization in Figure 1.

PathSim measures the similarity between entities over a given relationship [25]. For example, it may compute the similarity of two movies in a movie database based on their common actors. PathSim uses meta-walks to represent relationships between entities. For instance, the relationship between two movies in Figure 1b based on their common actors is expressed by [*film*,*actors*,*actor*,*actors*,*film*]. Let $p(e, f, d)$ be a set of walks of meta-walk p from entity e to entity f in database D. PathSim measures the similarity between e and f according to the input meta-walk p as $s(e, f) = \frac{2\times|p(e,f,D)|}{|p(e,e,D)|+|p(f,f,D)|}$. PathSim considers walks with and without consecutive forward and backward traverses from an entity to a node without value when it computes $s(e, f)$.



(a) DBLP-citation (b) SNAP

Figure 3: Fragments of two citation databases

PathSim may return different answers for the same queries over the same relationship on a database and its relationship reorganizations. Figure 3 shows fragments of DBLP from *dblp.uni-trier.de*, called DBLP-citation, and SNAP from
snap.stanford.edu that contain information about citations. Consider the meta-walk s = [*paper*, *cite*, *paper*, *cite*, *paper*] in Figure 3a, and its corresponding meta-walk s' = [*paper*, *paper*, *paper*] in Figure 3b. s has a walk between entities $p3$ and $p4$, x = [*paper*:p3, *cite*, *paper*:p4, *cite*, *paper*:p4]. But, there is no corresponding walk of meta-walk s' between $p3$ and $p4$ in Figure 3b. Hence, PathSim reports $p1$ to be more similar to $p2$ than $p3$ in Figure 3a, but considers $p1$ to be more similar to $p3$ than $p2$ in Figure 3b. PathSim returns different answers because it considers walks with consecutive forward and backward traverses from an entity to a node without value, such as x.

From here onward, we call a walk with consecutive forward and backward traverses from an entity to a node without value **informative**, and **non-informative** otherwise. As discussed in Section 4.1, non-informative walk either do not provide any information about the relationship between entities or their information can be represented by a shorter walk. Figure 3a and Figure 3b show that non-informative walks may be present in a database but be absent from its relationship-reorganizing transformations. Hence, if we modify PathSim so that it computes similarity scores using only informative walks, it will be representation independent under relationship-reorganizing transformations. Using Definition 4.2 and 4.3, we have the following theorem.

Theorem 4.5. *Let $T : \boldsymbol{L} \rightarrow \boldsymbol{K}$ be a relationship-reorganizing transformation and p be a meta-walk in $D \in \boldsymbol{L}$. There is a meta-walk r in $T(D)$ such that for each pair of entities e and f in D, we have $|p(e, f, D)| = |r(T(e), T(f), T(D))|$.*

Proof. Suppose p is maximal. According to Definition 4.3, there is a maximal meta-walk $T(p)$ in $T(D)$ s.t. $p \equiv T(p)$. Because there is a bijection that maps each informative walk of p to an informative walk of $T(p)$ with equal value, we have $|p(e, f, D)| = |T(p)(T(e), T(f), T(D))|$. If p is not maximal, according to Definition 4.2, we can find a maximal meta-walk p' in D that includes p s.t. $|p(e, f, D)| = |p'(e, f, D)|$. Using similar arguments to when p is maximal, we have that there exists a maximal meta-walk $T(p')$ in $T(D)$ s.t. $|T(p')(T(e), T(f), T(D))|$ $= |p'(e, f, D)|$. Hence, $|p(e, f, D)| = |T(p')(T(e), T(f), T(D))|$. □

Given entities e and f and a meta-walk p in database D and their corresponding entities and meta-walk in $T(D)$, $T(e)$, $T(f)$, and r, the numerator and denominator of $s(e, f)$ will be respectively equal to the numerator and denominator of $s(T(e), T(f))$. Hence, the modification of PathSim will return equal similarity scores for queries over a database and its relationship-reorganizing transformation. We call this extension of PathSim, *Robust-PathSim* (*R-PathSim*).

Let us discuss why we can modify PathSim to create a representation-independent algorithm and whether it is possible to extend other algorithms, such as RWR and SimRank, and make them representation independent. Relationship-reorganizing transformations do not add any relationship to or remove any relationship from a database. Because R-PathSim quantifies the amount of similarity separately for each type of relationship between two entities, it can return equal scores over a database and its relationship-reorganizing transformations. R-PathSim also leverages the concept of meta-walk to detect and ignore the spurious walks in each meta-walk that may not be present in some representations of the database. RWR and SimRank do not compute the similarity between entities based on a given relationship. One may define RWR or SimRank scores between two entities for a given meta-walk [25]. Also, we can modify RWR and SimRank to ignore the non-informative walks. Analyses of such extensions are interesting subjects of future work.

The computation of R-PathSim is similar to that of PathSim [25] with extra steps of detecting and ignoring non-informative walks. The *commuting matrix* of meta-walk $p = [l_1, \cdots, l_k]$ in database D is $M_p = A_{l_1 l_2} A_{l_2 l_3} \cdots A_{l_{k-1} l_k}$, where $A_{l_i l_j}$ is the adjacency matrix between nodes of labels l_i and l_j in D. Each entry $M_p(i,j)$ represents the the number of walks between entities $i \in l_i(D)$ and $j \in l_j(D)$. Given commuting matrix M_p, we can compute the PathSim score between i and j as $\frac{2M_p(i,j)}{M_p(i,i)+M_p(j,j)}$. However, R-PathSim uses only the informative walks. A meta-walk whose walks may not be informative is in the form of $p = [l_1, \cdots, l_i, x_{n_i}, \ldots, x_{m_i}, l_i, \ldots, l_k]$, $1 \leq i \leq k$ where l_i's are entity labels and $x_{n_i}, \cdots, x_{m_i}$ are labels of nodes without values. Meta-walk p may have non-informative walks because it contains meta-walks $s_i = [l_i, x_{n_i}, \ldots, x_{m_i}, l_i]$. Let M_{s_i} be the commuting matrix of s_i. The diagonal entries in M_{s_i} contain the number of non-informative walks of s_i. Let $M^d_{s_i}$ denote a diagonal matrix of M_{s_i}. Matrix $M_{s_i} - M^d_{s_i}$ contains the number of informative walks of s_i. To compute the number of informative walks of meta-walk p, we first find subwalks of p that start and end with same entity label and their remaining labels are non-entity labels. We call this set of meta-walks S and denote the rest of the subwalks of p R. The number of informative walks of p between each pair of entities in D is $M^i_p = \prod_{s \in S}(M_s - M^d_s) \prod_{r \in R} M_r$.

It may take a long time to compute the commuting matrix of a relatively long meta-walk in query time. Also, it is not feasible to precompute and store the commuting matrices for every possible meta-walk. PathSim precomputes commuting matrices for relatively short meta-walks. Then, PathSim concatenates them in the query time to get the commuting matrix of a longer meta-walk. This approach efficiently computes PathSim scores [25]. We follow the same method to compute R-PathSim scores efficiently.

Users may not know the structure of the database and cannot supply any input meta-walk. One may solve this problem by computing the (weighted) average of similarity scores over maximal meta-walks between entities [25]. Definition 4.3 provides that there is a bijection between all maximal meta-walks in a database and its relationship-reorganizing transformation. Also, Theorem 4.5 guarantees that R-PathSim returns equal scores for each maximal meta-walk over a database and its transformation. Hence, the combined similarity scores are equal in the original and transformed databases.

In order to find a set of maximal meta-walks, we first find a set of meta-walks in the database. Then we check if the meta-walks found are maximal or not. Algorithm 1 provides a framework on checking whether a given meta-walk is maximal. The underlying idea is that, if a meta-walk p is not maximal, there exists a meta-walk p' that includes p. That is, p' must contain an additional entity label to p. Using Definition 4.2, we check whether each walk of p is a subwalk of exactly one walk of p'. If there is p' that includes p, then p is not maximal. Otherwise, p is maximal. The running time of Algorithm 1 is $O(nd^3m)$ where n is the size of a given meta-walk p, d is the average degree of nodes, and m is the number of walks of p in the database.

For further optimization, if there are many meta-walks between the query node and the candidate answers in the database, one may save processing time by limiting the set of meta-walks over which the aggregated score is computed. One may do so by selecting the maximal meta-walks $p = rr^{-1}$, where r^{-1} is a meta-walk that is the reverse of r, such that r contains only distinct entity labels and only a given number of entity labels. Definition 4.3 guarantees that, for each maximal meta-walk r, there is exactly one maximal meta-walk r' in the transformed database such that $r \equiv r'$. Further, the number of entity labels of r and r' must be the same. That is, if p is used over the database, then $p' = r'r'^{-1}$ is also used over its transformation . Similar to Theorem 4.5, we have that the R-PathSim score computed using p over the database and using p' over its transformation are equal. Therefore, the aggregated R-PathSim score computed over these sets are equal across the original database and its transformations.

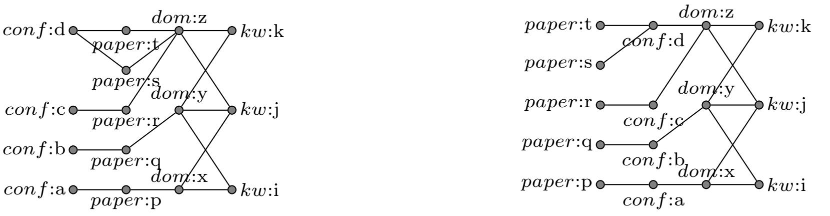
Algorithm 1: Check a meta-walk if it is maximal

```
Input: Database $D = (V, E, L, A)$, Meta-walk $p = [l_1, ..., l_n]$
Output: ACCEPT if $p$ is maximal, or REJECT if $p$ is not maximal
foreach $i = 2...n - 1$ do
    $S_i \leftarrow$ set of all meta-walks $[l_i, l]$ or $[l_i, l', l]$ in $D$ where $l$ is an entity label and $l'$ is not an entity label
    foreach $r \in S_i$ do
        foreach $w = [v_1, ..., v_n] \in p(D)$ do
            if there exists no walk or more than two walks in $r(D)$ from $v_i$ then
                /* Assume $p' = [l_1, ..., l_i]rr^{-1}[l_i, ..., l_n]$ where $p \sqsubset p'$. $p'$ does not include $p$. */
                Go to process next $r \in S_i$
            end
        end
        /* Each walk of $p$ is a subwalk of exactly one walk of $p'$. Hence, $p'$ includes $p$. */
        return REJECT
    end
end
return ACCEPT
```

5 Entity Rearrangement

5.1 Entity-Rearranging Transformation

Different databases may represent the same relationship between a set of entities by connecting them using different sets of edges. Consider Figure 4 that shows the original and an alternative representation for Microsoft Academic Search (*academic.research.microsoft.com*) (MAS for short) data. Both databases contain entities of semantic types paper, conference, domain, and keyword, which are labeled as *paper*, *conf*, *dom*, and *kw*, respectively. The domains of papers and conferences show their areas, e.g., *database* and *data mining*. The keyword entities contain the keywords of domains, e.g., *indexing* for *database* domain. Each paper is published in only one conference and each conference belongs to only one domain. The database in Figure 4a expresses the relationship between a paper and its conference and domain by connects each paper to both its conference and its domain. On the other hand, the database in Figure 4b represents the same relationship by connecting each paper to its conference and connecting each conference to its domain. We call this representational shift that rearranges entities in a database an entity-rearranging transformation.



(a) Original representation

(b) Alternative representation

Figure 4: Fragments of some representations for MAS data with FDs $paper \rightarrow conf$ and $conf \rightarrow dom$.

Because each paper in Figure 4b has only one conference and each conference has only one domain, we can switch the relative position of paper and conference and get Figure 4a without losing or adding any relationships to the ones represented in Figure 4b. Assume that a paper can be published in multiple conferences from different domains in a database that follows the representation of Figure 4b. If we rearrange the positions of papers, conferences, and domains in this database according to the representation in Figure 4a, each conference of a paper will be connected to all domains of every conference in which the paper is published. Hence, we will add new relationships between conferences and domain that are not available in the original database. Also, we will not be able to recover the original set of relationships between a conference of its domain in the original database. Hence, an entity-rearranging transformation preserves the information of a database if certain entities in the database satisfy some dependencies. The following definition formalizes these dependencies. Let $l(D)$ denote all nodes in database D with label l.

Definition 5.1. *Given meta-walk p =[l_1, ..., l_n] in the set of databases* $\boldsymbol{L}$*,* $\boldsymbol{L}$ *satisfies* **functional dependency** *(FD)* $l_1 \xrightarrow{p} l_n$ *iff for every* $D \in \boldsymbol{L}$ *if walks [e,...,f] and [e,...,g] of meta-walk p are in D, then* $f = g$.

For example, the FDs in Figure 4a are *paper* $\xrightarrow{p_1}$ *conf*, *paper* $\xrightarrow{p_2}$ *dom*, and *conf* $\xrightarrow{p_3}$ *dom*, where $p_1 = [paper, conf]$, $p_2 = [paper, conf, dom]$, and $p_3 = [conf, dom]$. Given meta-walk $p = [l_1, l_2]$, we write the FD $l_1 \xrightarrow{p} l_2$ as $l_1 \rightarrow l_2$ for brevity.

Intuitively, an entity-rearranging transformation should preserve the label and values of entities, the relationships between entities, and the FDs of a database to preserve its information. For example, there is a bijection between entities in Figure 4a and Figure 4b that preserve their labels and values. Moreover, if there is not any FD between some entities, an entity-rearranging transformation must not rearrange them. In other words, if we have edge (e, f) in database D and there is no FD between e and f, an entity-rearranging transformation T must map e and f to entities $T(e)$ and $T(f)$ in $T(D)$ with edge $(T(e), T(f))$. Similarly, if there is no edge between the aforementioned entities in D, there must not be any edge between them in $T(D)$. Furthermore, the transformed database must satisfy essentially the same FDs as the original database. That is, if there is an FD between entities of semantic types l_1 and l_2 in the original database, there must be an FD between the entities of l_1 and l_2 in the transformed database. Otherwise, as explained in the preceding paragraph, the transformation may introduce spurious relationships between entities. However, the corresponding FDs in the original and transformed databases may be represented using different meta-walks. For instance, FD *conf* $\xrightarrow{[conf, paper, dom]}$ *dom* in Figure 4a is mapped to *conf* $\rightarrow$ *dom* in Figure 4b. The following definition formalizes the aforementioned intuitions. Let $F_{\mathbf{L}}$ denote the set of FDs satisfied by the set of databases $\mathbf{L}$.

Definition 5.2. *A transformation $T : \boldsymbol{L} \rightarrow \boldsymbol{K}$ that maps database $D = (V_D, E_D, \mathcal{L}, \mathcal{A}_D)$ to database $T(D) = (V_{T(D)}, E_{T(D)}, \mathcal{K}, \mathcal{A}_{T(D)})$ is* **entity rearranging** *iff there is a bijection $M : V_D \rightarrow V_{T(D)}$ such that*

- *for all $v \in V_D$, $\mathcal{L}(v) = \mathcal{K}(M(v))$ and if v is an entity, $\mathcal{A}_D(v) = \mathcal{A}_{T(D)}(M(v))$.*
- *for all $(u, v) \in V_D$ where neither $\mathcal{L}(u) \rightarrow \mathcal{L}(v)$ nor $\mathcal{L}(v) \rightarrow \mathcal{L}(u)$ are in $F_{\boldsymbol{L}}$, we have $(u, v) \in E_D$ iff $(M(u), M(v)) \in E_{T(D)}$.*
- *there is a bijection $N : F_{\boldsymbol{L}} \rightarrow F_{\boldsymbol{K}}$ such that if $N(l_1 \xrightarrow{p} l_2) = l_1 \xrightarrow{p'} l_2$, for all entities $e, f \in V_D$, $p(e, f, D)$ is empty iff $p'(M(e), M(f), T(D))$ is empty.*

Using Definitions 5.1 and Definition 5.2, we have the following.

Theorem 5.3. *Each entity-rearranging transformation is similarity preserving.*

Proof. Let $T : \mathbf{L} \rightarrow \mathbf{K}$ be an entity-rearranging transformation. For each $D = (V, E, \mathcal{L}, \mathcal{A}) \in \mathbf{L}$, let M be the bijection that T establishes between the set of nodes in D and the set of nodes in $T(D)$ according to Definition 5.2. Let $N : F_L \rightarrow F_K$ be the bijection that T establishes s.t. if $N(l_1 \xrightarrow{p} l_2) = l_1 \xrightarrow{p'} l_2$, for all entities $e, f \in V_D$, $p(e, f, D) = \emptyset$ iff $p'(M(e), M(f), T(D)) = \emptyset$. Let M^{-1} and N^{-1} be the inverses of M and N, respectively. Let us define a transformation T' from $\mathbf{K}$ to $\mathbf{L}$ as follows. For each $D' = (V', E', \mathcal{K}, \mathcal{A}') \in \mathbf{K}$, (1) $\forall v \in V'$, $\mathcal{K}(v) = \mathcal{L}(M^{-1}(v))$ and $\mathcal{A}'(v) = \mathcal{A}(M^{-1}(v))$, (2) $\forall u, v \in V'$, if $\mathcal{K}(u) \rightarrow \mathcal{K}(v)$, $\mathcal{K}(v) \rightarrow \mathcal{K}(u) \notin F_K$, then $(u, v) \in V'$ iff $(M^{-1}(u), M^{-1}(v)) \in V$, and (3) N^{-1} bijectively maps $l_1 \xrightarrow{p'} l_2 \in F_K$ to $l_1 \xrightarrow{p} l_2 \in F_L$ s.t. $\forall e, f \in V'$, $p'(e, f, D') = \emptyset$ iff $p(M^{-1}(e), M^{-1}(f), T'(D')) = \emptyset$. Next, we show that $T'(T(D))$ and D are the same database.

Let V_D, $V_{T(D)}$ and $V_{T'(T(D))}$ denote the sets of nodes in D, $T(D)$ and $T'(T(D))$, respectively. Using Definition 5.2 and the construction of T', we construct a bijection $H : V_D \rightarrow V_{T'(T(D))}$ s.t. $H = M^{-1} \circ M$. For each $v \in V_D$, the labels and values of v and $H(v)$ are the same. Thus, $V_D = V_{T'(T(D))}$. Consider for each edge $e = (u, v)$ in D. If $\mathcal{L}(u) \rightarrow \mathcal{L}(v)$,$\mathcal{L}(v) \rightarrow \mathcal{L}(u) \notin F_L$, then $\mathcal{K}(M(u)) \rightarrow \mathcal{K}(M(v))$, $\mathcal{K}(M(v)) \rightarrow \mathcal{K}(M(u)) \notin F_K$, and $\mathcal{L}(M^{-1}(M(u))) \rightarrow \mathcal{L}(M^{-1}(M(v)))$, $\mathcal{L}(M^{-1}(M(v))) \rightarrow \mathcal{L}(M^{-1}(M(u))) \notin F_L$. Using Definition 5.2 and the construction of T', we have that (u, v) exists in D iff $(H(u), H(v)) = (M^{-1}(M(u)), M^{-1}(M(v))$ exists in $T'(T(D))$. Otherwise, without losing generality, assume $\mathcal{L}(u) \rightarrow \mathcal{L}(v) \in F_L$. Let $p = [\mathcal{L}(u), \mathcal{L}(v)]$. N bijectively maps $\mathcal{L}(u) \rightarrow \mathcal{L}(v) \in F_L$ to $\mathcal{L}(u) \xrightarrow{p'} \mathcal{L}(v) \in F_K$ for some meta-walk p'. Also, N^{-1} bijectively maps $\mathcal{L}(u) \xrightarrow{p'} \mathcal{L}(v) \in F_K$ to $\mathcal{L}(u) \rightarrow \mathcal{L}(v) \in F_L$. N guarantees that, $\forall u, v \in V_D$, $p(u, v, D) = \emptyset$ iff $p'(M(u), M(v), T(D)) = \emptyset$. Also, N^{-1} guarantees that, $\forall u', v' \in V_{T(D)}$, $p'(u', v', T(D)) = \emptyset$ iff $p(M^{-1}(u'), M^{-1}(v'), T'(T(D))) = \emptyset$. That is, $\forall u, v \in V_D$, $p(u, v, D) = \emptyset$ iff $p(H(u), H(v), T'(T(D))) = \emptyset$. Because we assume that our data graph is simple, if $p(u, v, D)$ is not empty, then there is exactly one walk $[u, v]$ in $p(u, v, D)$ which is a walk along an edge (u, v). Similarly, if $p(H(u), H(v), T'(T(D)))$ is not empty, then an edge $(H(u), H(v))$ exists in $T'(T(D))$. That is, (u, v) exists in D iff $(H(u), H(v))$ exists in $T'(T(D))$. Thus, D and $T'(T(D))$ are the same, and so T is invertible.

Using the first condition in Definition 5.2, each entity-rearranging transformation is entity preserving. Therefore, every entity-rearranging transformation is similarity preserving. □

Entity-rearranging transformations resemble (de)normalization in relational and tree-shaped XML databases [1, 3]. They, however, modify the connections between entities in the database instead of removing duplicates and are defined over graph databases that follow less restrictive schemas than relational schemas or DTDs.

5.2 Extension of R-PathSim

Because entity-rearranging transformations modify the topology the database, RWR and SimRank are not robust under these transformations. For example, consider the entity-rearranging transformation between Figure 4a and Figure 4b. RWR and SimRank find *paper*:p to be more similar to *paper*:r than *paper*:t in Figure 4b. However, they find *paper*:p to be more similar to *paper*:t than *paper*:r in Figure 4a. R-PathSim and PathSim are also not robust under entity-rearranging transformations. A user may like to find conferences similar to *conf*:b based on their common keywords using meta-walk p_1 = [*conf*, *dom*, *kw*, *dom*, *conf*] in Figure 4b. R-PathSim finds *conf*:a and *conf*:c equally similar to *conf*:b. The meta-walk that represents the closets relationship to p_1 in Figure 4a is p_2 = [*conf*, *paper*, *dom*, *kw*, *dom*, *paper*, *conf*]. However, using meta-walk p_2, R-PathSim finds *conf*:a more similar to *conf*:b than *conf*:c in Figure 4a.

We observe that meta-walk p_2 does not exactly represent the same information as meta-walk p_1 because p_2 contains additional entity labels, i.e., *paper*. Hence, a walk in p_1 may correspond to several walks in p_2. This causes R-PathSim to produce different rankings for the same query over Figures 4a and 4b. To return the same answers over Figure 4a and 4b, one may look for a structure in Figure 4a that represents exactly the same information that p_1 expresses in Figure 4b. Every walk of p_1 represents the fact that a conference belongs to a certain domain and does not contain any information about the number of papers published in the conference. Hence, we extend the definition of meta-walk to ignore the number of occurrences of certain entities in a walk. For example, we define meta-walk p_3 in Figure 4a whose walks express the fact that entities of labels *conf* and *dom* are connected through *paper* entities without any regard to the number of papers between them. This meta-walk treats all walks between each pair of entities of label *conf* and *dom* through *paper* entities as a single walk. We show p_3 as [*conf*, $\overline{\textit{paper}}$, *dom*, *kw*, *dom*, $\overline{\textit{paper}}$, *conf*]. We call $\overline{\textit{paper}}$ a $*$-*label* in p_3. Using a $*$-label in the meta-walk indicates that the user is interested in whether a connection between entities in the meta-walk exists. Meta-walk p_3 has the same number of walks in Figure 4a as p_1 has in Figure 4b. Hence, R-PathSim will deliver the same ranking for query *conf*:b over Figure 4a using meta-walk p_3 and Figure 4b using meta-walk p_1.

Furthermore, we may have to use a more complex meta-walk in a database to express the same information as a simpler meta-walk in the entity-rearranging transformation of the database. For instance, a user may like to find similar conferences based on the meta-walk $p2$ in Figure 4a. However, she must use a more complex meta-walk [*conf*, *paper*, *conf*, *dom*, *kw*, *dom*, *conf*, *paper*, *conf*] to obtain the same results in Figure 4b. Instead of stopping at the candidate answer label, this meta-walk goes beyond and traverses back to the candidate answer label. We call this type of meta-walks *meta-walks with repeated entities*.

Hence, a relationship between the same set of entities may be represented by normal meta-walks, as defined in Section 4 in a database, but using meta-walks with $*$-label or repeated entities on its entity-rearranging transformations. To be robust over entity-rearranging transformations, R-PathSim should consider meta-walks with $*$-label and repeated entities in addition to the meta-walks defined in Section 4. Thus, we extend R-PathSim to consider these types of meta-walks. Nevertheless, if we allow $*$-label for every label in each meta-walk, R-PathSim has precompute the commuting matrices for a large number of meta-walks. Hence, we would like to identify a minimal set of meta-walks with $*$-label(s) that capture all relationships in a database and R-PathSim can use them to deliver the same results over the database and its entity rearrangements. First, according to Definition 5.2, if neither meta-walk p nor any of its subwalks is a meta-walk for any FD over a database D, there is a meta-walk r over the entity-rearranging transformation of D, $T(D)$ such that r and p have equal number of walks in D and $T(D)$, respectively. Thus, R-PathSim score of entities over these meta-walks are equal over D and $T(D)$. Hence, we assign $*$-label only on a meta-walk that determines some FD in a database. Second, consider meta-walk $s = [l_1, \ldots, l_k]$ in database D where $l_1 \rightarrow l_2$, $l_2 \rightarrow l_3$, $\ldots$, $l_{k-1} \rightarrow l_k$ hold in D. If there is a walk from entity e of label l_1 and entity f of label l_k in D, there will be exactly one walk between e and f in D because of the FDs over s. Hence, every meta-walk created by setting some of the labels in s to $*$-label have the same number of walks in D as s has. Intuitively, setting some labels in s to $*$-labels will not express any new useful relationship between entities in s. Moreover, because R-PathSim returns the same similarity score for two entities using s and its modifications, we will consider only s.

Next, we prove that the aforementioned extension of R-PathSim is representation independent over entity-rearranging transformations. Let $\mathbf{L}$ be a set of databases whose set of labels is L. Let $S \subseteq L$ be a set of labels in a database. We define a binary relation $\prec_S$ between labels l_1 and l_2 in L where $l_1 \prec_S l_2$ iff there is a meta-walk p whose labels exists in S such that $l_1 \xrightarrow{p} l_2 \in F_{\mathbf{L}}$. We define S to be a **chain** iff $\prec_S$ is a total order over S. S is a **maximal chain** iff there is no $R \subseteq L$ such that $S \subsetneq R$ and R is a chain. For instance, because we have *paper* $\rightarrow$ *conf*, *paper* $\rightarrow$ *dom* and *conf* $\xrightarrow{[conf, paper, dom]}$ *dom* in Figure 4a, {*paper*,*conf*,*dom*} is a maximal chain. By the abuse of notation, we let F_S denote the set of FDs in $\mathbf{L}$ whose labels are in S. In this paper, we focus on a set of databases whose sets of maximal chains are mutually exclusive. MAS databases whose fragments are shown in Figure 4a and 4b are examples of such databases.

Theorem 5.4. *Given an entity-rearranging transformation $T : \boldsymbol{L} \rightarrow \boldsymbol{K}$, for all $D \in \boldsymbol{L}$, T bijectively maps each meta-walk p in D to a meta-walk r in $T(D)$ such that for all entities e and f in D, $|p(e, f, D)| = |r(T(e), T(f), T(D))|$.*

Proof. Let us define *internal* labels of a meta-walk $p = [l_1, ..., l_n]$ as labels $l_2, ...l_{n-1}$. Given a meta-walk p in D, one can write p as a concatenation of meta-walks $s_1...s_m$ where m is the smallest value s.t. each s_i, $i = 1...m$, satisfies exactly one of the following conditions: (1) s_i is not a meta-walk of any FD in $\mathbf{L}$, (2) all internal labels of s_i are $*$-labels, or (3) $s_i = [l'_1, ..., l'_k]$ where $l'_1 \rightarrow l'_2, ..., l'_{k-1} \rightarrow l'_k \in F_L$ (or $l'_k \rightarrow l'_{k-1}, ..., l'_2 \rightarrow l'_1 \in F_L$). Clearly, no s_i satisfies both conditions (1) and (3), or both (2) and (3). Because we set the labels of only meta-walks used in an FD to $*$-labels, no s_i satisfies both conditions (1) and (2). Suppose there are more then one concatenation of p that satisfies the aforementioned conditions. Without losing generality, assume $p = s_1s_2 = s'_1s'_2$ where $s_1 = [l_1, ..., l_k]$, $s_2 = [l_k, ..., l_n]$. If s_1 satisfies condition (1), then s_2 satisfies either condition (2) or (3). Otherwise, $m = 2$ is not the smallest number that satisfies the aforementioned concatenation for p. Clearly, s'_1 cannot satisfies condition (2) or (3). Also, $[l_1, ..., l_k, ..., l_{k'}]$, $k < k' < n$, cannot satisfies condition (1). Hence, $s_1 = s'_1$ and $s_2 = s'_2$. If s_1 satisfies condition (2), then any contiguous proper subwalk of s_1 and a walk $[l_1, ..., l_k, ..., l_{k'}]$, $k < k' < n$, cannot satisfies any of the aforementioned conditions. Hence, $s'_1 = s_1$ and $s'_2 = s_2$. If s_1 satisfies condition (3) and s_2 either satisfies condition (1) or (2), then any contiguous proper subwalk of s_2 and a walk $[l_{k'}, ..., l_k, ..., l_n]$, $1 < k' < k$, cannot satisfies any of the aforementioned conditions. Thus, $s'_2 = s_2$, and so $s'_1 = s_1$. If s_1 and s_2 satisfy condition (3), then either (a) $l_1 \rightarrow l_2, ..., l_{k-1} \rightarrow l_k$ and $l_n \rightarrow l_{n-1}, ..., l_{k+1} \rightarrow l_k$, or (b) $l_k \rightarrow l_{k-1}, ..., l_2 \rightarrow l_1$ and $l_k \rightarrow l_{k+1}, ..., l_{n-1} \rightarrow l_n$. Thus, any contiguous proper subwalk of s_1 and a walk $[l_1, ..., l_k, ..., l_{k'}]$, $k < k' < n$, cannot satisfies any of the aforementioned conditions. Hence, $s'_1 = s_1$ and $s'_2 = s_2$. Therefore, there is exactly one concatenation of p that satisfies the aforementioned conditions.

Let $s_i = [l'_1, ..., l'_k]$. Suppose s_i satisfies condition (1). Using Definition 5.2, there is a bijection between walks of s_i and walks of $T(s_i)$ s.t. $|s_i(e, f, D)| = |T(s_i)(T(e), T(f), T(D))|$. Suppose s_i satisfies condition (2). Because we set the labels of only meta-walks used in an FD to $*$-labels, we have $l'_1 \xrightarrow{s_i} l'_k$ (or $l'_1 \xleftarrow{s_i} l'_k$). By Definition 5.2, T bijectively maps $l'_1 \xrightarrow{s_i} l'_k$ to some $l'_1 \xrightarrow{r_i} l'_k$ in $F_{\mathbf{K}}$ s.t. $s_i(e, f, D) = \emptyset$ iff $r_i(T(e), T(f), T(D)) = \emptyset$. If $s_i(e, f, D) \neq \emptyset$, then $|s_i(e, f, D)| = 1$ and $r_i(T(e), T(f), T(D)) \neq \emptyset$. Let r^*_i be obtained by changing all internal labels of r_i to $*$-labels. If $r_i = [l'_1, l''_1, ..., l''_{k'}, l'_k]$ and $l'_1 \rightarrow l''_1, ..., l''_{k'} \rightarrow l'_k$, then $r^*_i = r_i$. If $r_i(T(e), T(f), T(D)) \neq \emptyset$, then $|r^*_i(T(e), T(f), T(D))| = 1$. Hence, $|s_i(e, f, D)| = |r^*_i(T(e), T(f), T(D))|$. Suppose s_i satisfies condition (3). If $s_i(e, f, D) \neq \emptyset$, then $|s_i(e, f, D)| = 1$. Similar to the case where s_i satisfies condition (2), we prove that T bijectively maps s_i to r^*_i s.t. $|s_i(e, f, D)| = |r^*_i(T(e), T(f), T(D))|$.

The end node of each walk in s_i is the start node of a walk s_{i+1}. Hence, by Definition 5.2, the end node of each walk of $T(s_i)$ is the start node of a walk of $T(s_{i+1})$. Let r be the meta-walk created by concatenating $T(s_i)$'s. Each walk of r is a concatenation of walks of $T(s_i)$ in $T(D)$. □

Similar to Section 4.2, one may compute a single similarity between a pair of entities by computing the average of R-PathSim scores over all meta-walks between the pair of entities. Theorem 5.4 guarantees that the aggregated similarity scores for each pair of entities and their mapping over entity-rearranging transformations are equal. We use the same methods discussed in Section 4.2 to precompute and compute the score of meta-walks with $*$-labels and repeated entities. Our results here introduce a new method to make a similarity search algorithm representation independent. Because the same relationship may be expressed in several forms over different representations of the same data, the algorithm should consider more varieties of relationships.

Theorem 5.4 guarantees that aggregated R-PathSim computed over all meta-walks including ones that use $*$-label or repeated nodes returns the same ranked list of answers as aggregated R-PathSim computed over those meta-walks in the transformed databases under entity rearranging. Since the set of all meta-walks in a database is infinite, it is impractical to compute R-PathSim over all meta-walks in a databases. Limiting the size of meta-walks to be computed as in Section 4.2 helps reducing the number of meta-walks to be computed over; however, this solution does not guarantee that the results over the database and its entity-rearranging transformed database are the same. Suppose a database in Figure 4a is a fragment of a database D that contains labels that are not exists in Figure 4, and those labels are not part of any functional dependencies. Let D be transformed to a database E by the entity-rearranging transformation used in transforming Figure 4a to Figure 4b. Consider that a mapping to a meta-walk $p_1 =$ [*conf*, *paper*, *dom*, *paper*, *conf*] in D is $p_2 =$ [*conf*, *paper*, *conf*, *dom*, *conf*, *paper*, *conf*] in E. Assume the limit of meta-walk is a large number N. There exists a meta-walk $p = rp_1s$ of size N in D where r and s do not contain any edge that is part of any FD in D. The mapping that follows Theorem 5.4 in E is $p' = rp_2s$ in which the size of p' is more than p. Following this example, we can argue that no matter how big the size limit of meta-walks is, there is an entity-rearranging transformation such that there is a meta-walk whose mapping meta-walk that follows Theorem 5.4 has the size that exceeds the given limit, or vice versa.

For aggregated R-PathSim to be representation independent while being accurate and efficient, we should only compute aggregated R-PathSim over a subset of meta-walks in a database. Given a set E of entity labels in a database with a set F of FDs, we propose Algorithm 3 that finds a subset of meta-walks whose labels exists in E or appears in some chain in F.

Algorithm 3 first finds a subset S of meta-walks without any repeated labels that contains only entity labels from E or some chain in the database. Then it adds more meta-walks to the set S by modifying each meta-walk in S if its edge is used in an FD in some chain. While keeping the part of each meta-walk whose edges does not appear as any FD intact, the algorithm modifies other parts by either extending them to reach the label that determines other labels in the chain, or otherwise mark the labels in these parts as $*$-labels. Finally, each meta-walk S is concatenated with its own reverse so that the meta-walk starts from the query label and end at the same query label. Suppose the maximum number of distinct labels that are adjacent to a node is d. Let $|L_F|$ denote the set of labels of FDs in F. Since S is constructed such that no labels are repeated in each meta-walk, then S contains at most $O(d!)$ meta-walks before the modification. In each meta-walk, there can be at most $O(|L_F|)$ contiguous subwalks that involves an FD in some chain. Thus, there are $O(2^{|L_F|})$ possible modifications for each meta-walk. Therefore, the size of the returned subset of meta-walks is $O(d!2^{|L_F|})$. Follows Proposition 5.6, the R-PathSim score computed over the set obtained by Algorithm 2 is the same as the R-PathSim score computed over the set of meta-walks obtained by this algorithm over the transformed database.

Lemma 5.5. *Let L be a chain in a database. Let $l_1 \xrightarrow{p} l_n \in F_L$ where $p = [l_1, ..., l_n]$, $l_1, ..., l_n \in L$. For all $i = 2...n-1$, there is no $j \in \{1, ..., i-1\}$ and $k \in \{i+1, ..., n\}$ such that $l_j \xrightarrow{[l_j,...,l_i]} l_i, l_k \xrightarrow{[l_k,...,l_i]} l_i \in F_L$.*

Proof. Given a database $D \in \mathbf{L}$. Suppose there exists some $i \in \{2, ..., n-1\}$ where there are $j \in \{1, ..., i-1\}$ and $k \in \{i+1, ..., n\}$ s.t. $l_j \xrightarrow{[l_j,...,l_i]} l_i, l_k \xrightarrow{[l_k,...,l_i]} l_i \in F_L$. Let $p_1 = [l_j, ..., l_i]$ and $p_2 = [l_k, ..., l_i]$. Then there are entities e, f_1, $f_2 \neq f_1$ and g whose labels are l_1, l_n, l_n and l_i, respectively, s.t. $p_1(e, g, D) \neq \emptyset$, $p_2(f_1, g, D) \neq \emptyset$ and $p_2(f_2, g, D) \neq \emptyset$. Since $p = p_1 p_2$, $p(e, f_1, D) \neq \emptyset$ and $p(e, f_2, D) \neq \emptyset$. That is, $l_1 \xrightarrow{p} l_n$ does not hold in F_L which is contradiction. □

Proposition 5.6. *Let $T : \mathbf{L} \to \mathbf{K}$ be an entity-rearranging transformation. Given a query q in a database $D \in \mathbf{L}$, let S_D and $S_{T(D)}$ be sets of meta-walks returned by Algorithm 2 using the same inputs over D and $T(D)$, respectively. The aggregated R-PathSim score over S_D for each candidate answer in D, and the aggregated R-PathSim score over $S_T(D)$ for the same candidate answer in $T(D)$ are the same.*

Proof. Consider each meta-walk $m \in S_D$. We have that $m = pp^{-1}$ for some meta-walk p that starts with label l in D. Similarly, for each meta-walk $m' \in S_{T(D)}$, $m' = p'p'^{-1}$ for some meta-walk p' that starts with label l in $T(D)$. By the construction of p by Algorithm 2, we can write p as $p = p_1...p_k$ where k is the smallest s.t. each p_i, $i = 1...k$, follows either (1) for every edge (u, v) of p_i, $u \to v$, $v \to u \notin F_D$, or (2) there exists a maximal chain C in $T(D)$ s.t. for every edge (u, v) of p_i, $u \to v$ or $v \to u$ exists in C. Without losing generality, we will show that there is a bijective mapping M between p_i in D and p'_i in $T(D)$ s.t. for every pair of entities e and f, $|p(e, f, D)| = |p'(e, f, T(D))|$. By Definition 5.2, each p_i that follows condition (1) exists in both D and $T(D)$. Further, each walk of p_i exists in both D and $T(D)$. Thus, for every pair of entities e and f, $|p_i(e, f, D)| = |p_i(e, f, T(D))|$. For case (2), assume $p_i = [l_1, ..., l_n]$ where $l_1, ..., l_n$ belongs to some maximal chain C in D. Let l_o be the smallest in C under $\prec_C$. We prove p_i for each of the following cases. (Case 1) Suppose $l_1 = l_o$ (or $l_n = l_o$). By Definition 5.2 and because we assume that sets of maximal chains in a databases are mutually exclusive, there exists exactly one meta-walk p'_i s.t. every label of p_i exists in C. By Lemma 5.5, we have that $l_1 \to l_2, ..., l_{n-1} \to l_n$ in D. For every pair of entities e and f, $|p_i(e, f, D)| = 1$. Because p_i must also starts and ends with the same labels as those of p_i, using similar arguments, then $|p'_i(e, f, T(D))| = |p_i(e, f, D)|$. (Case 2) Suppose $l_i \neq l_o$ for any $i = 1...n$, or $l_2, ..., l_{n-1}$ are $*$-labels. By Lemma 5.5, we have that $l_1 \to l_2, ..., l_{n-1} \to l_n$ (or $l_1 \leftarrow l_2, ..., l_{n-1} \leftarrow l_n$) in D. Hence, for every pair of entities e and f, $|p_i(e, f, D)| = 1$. Definition 5.2 bijectively maps $l_1 \xrightarrow{p_i} l_n$ in D to $l_1 \xrightarrow{p'_i} l_n$ where every labels of p'_i belongs to C. If l_o does not appear in p'_i, then using similar argument, $|p'_i(e, f, T(D))| = 1$ for every pair of entities e and f. Otherwise, the algorithm marks all labels except the first and the last in p'_i as $*$-labels, then $|p'_i(e, f, T(D))| = 1$. Therefore, $|p'_i(e, f, T(D))| = |p_i(e, f, D)|$. (Case 3) Suppose $l_j = l_o$ for some $j = 1...n$. By Lemma 5.5, we have that $l_1 \leftarrow l_2, ..., l_{j-1} \leftarrow l_j$, $l_j \to l_{j+1}, ..., l_{n-1} \to l_n$ in D. By definition of FD, $|p_i(e, f, D)|$ equals to the number of entities g of labels l_o that exists in walk of p_i from e to f in D. That is, $|p_i(e, f, D)| = \sum_g |[l_1, ..., l_j](e, g, D)| =$

Algorithm 2: MetaWalkFinders

```
Input: Database $D$, entity label $l$, set of entity labels $L$, integer $R$
Output: Subset $S$ of meta-walks in $D$ whose labels starting and ending with $l$
$S \leftarrow \{\}$
$C \leftarrow$ set of maximal chains in $D$
$L_C \leftarrow$ set of entity labels that exists in $C$
$L_{all} \leftarrow L \cup L_C$
$P \leftarrow$ set of meta-walks in $D$ whose occurrence of each entity label is at most $R$ that start with label $l$
and contain only entity labels from $L_{all}$
$M \leftarrow \emptyset$
foreach $l_1, l_2 \in L_C$ do
    $M[l_1][l_2] \leftarrow MetaWalksFinderFromChain(F, l_1, l_2)$
    /* By the construction of $P$, $l_1 \neq l_2$ */
end
foreach $p'' \in P$ do
    $S' \leftarrow \{[l]\}$
    Construct an ordered list $Parts = (p_1, ..., p_k)$ s.t. $p'' = p_1 ... p_k$ where $k$ is the smallest such that each
    $p_i$, $i = 1...k$, is either a meta-walk whose edges are not used in any FD or a meta-walk whose labels
    exists in a single FD in $F$.
    foreach $p' \in Parts$ do
        if every edge $(u, v)$ in $p'$, $u \rightarrow v$ does not appear in any chain in $F$ then
            /* Keep partition $p'$ of $p''$ whose edges are not used in any FD as is. */
            foreach $p \in S'$ do
                Remove $p$ from $S'$
                Add $pp'$ to $S'$
            end
        else
            /* Replace the partition of $p''$ whose edges are used in some FD by using its maximal chain */
            $l_1 \leftarrow$ first label of $p'$
            $l_2 \leftarrow$ last label of $p'$
            foreach $p \in S'$ do
                Remove $p$ from $S'$
                foreach $p''' \in M[l_1][l_2]$ do
                    Add $pp'''$ to $S'$
                end
            end
        end
    end
    $S \leftarrow S \cup S'$
end
/* Concatenate these meta-walks with its reverse so that each meta-walk visits label $l$ at the end */
foreach $p \in S$ do
    Replace $p$ with $pp^{-1}$
end
return $S$
```

$\sum_g |[l_j, ..., l_n](g, f, D)|$. Because $l_1, l_n \in C$, Definition 5.2 bijectively maps the FD between l_1 and l_n in D to $l_1 \xrightarrow{r} l_n$ in $T(D)$ for some meta-walk $r = [l'_1, ..., l'_{n'}]$ whose labels $l'_1, ..., l'_{n'}$ are in C, and $l'_1 = l_1$ and $l'_{n'} = l_n$. If $l'_{j'} = l_o$ for some $j' = 1...n''$, by using Lemma 5.5, we have that $l'_1 \leftarrow l'_2, ..., l'_{j'-1} \leftarrow l'_{j'}$, $l'_{j'} \rightarrow l'_{j'+1}, ..., l_{n'-1} \rightarrow l'_{n'}$ in D. That is, $|r(e, f, T(D))| = \sum_g |[l'_1, ..., l'_j](e, g, T(D))| = \sum_g |[l'_j, ..., l'_n](g, f, T(D))|$. Using definition of FD and Definition 5.2, $\sum_g |[l'_1, ..., l'_j](e, g, T(D))| = \sum_g |[l_1, ..., l_j](e, g, D)|$. Let $p'_i = r$, we have $|p'_i(e, f, T(D))| = |p_i(e, f, D)|$. Otherwise, there is no $j' = 1...n'$ s.t. $l_{j'} = l_o$. By Lemma 5.5, we have that $l'_1 \rightarrow l'_2, ..., l'_{n'-1} \rightarrow l'_{n'}$ (or $l'_1 \leftarrow l'_2, ..., l'_{n'-1} \leftarrow l'_{n'}$) in $T(D)$. Because sets of maximal chains are mutually exclusive, there exists exactly one meta-walk $s = [l''_1, ..., l''_{n''}]$ whose labels are in $C \setminus \{l'_1, ..., l'_{n'}\}$ s.t. $l_o \xrightarrow{s} l'_1$ where $l''_1 = l_o$ and $l''_{n''} = l_1$. Further, $l''_1 \rightarrow l''_2, ..., l''_{n''-1} \rightarrow l''_{n'}$ (or $l''_1 \leftarrow l''_2, ..., l''_{n''-1} \leftarrow l''_{n''}$). The algorithm constructs

Algorithm 3: MetaWalkFindersFromChain

```
Input: Set of maximal chains $F$, labels $L_1, L_2$, $L_1 \neq L_2$
Output: Set $M$ of meta-walks from $l_1$ to $l_2$ whose labels are from the maximal chain
$M \leftarrow \emptyset$
Find $f = \{l_1 \xrightarrow{r_1} l_2, l_2 \xrightarrow{r_2} l_3, ..., l_{n-1} \xrightarrow{r_{n-1}} l_n\} \in F$ s.t. $L_1 = l_j$ and $L_2 = l_k$, for some $j, k = 1...n$
/* There is at most one such $f$ because sets of maximal chains in a database are mutually exclusive. */
if $f$ exists then
    if $j > k$ then
        Swap labels between $l_j$ and $l_k$
        $swap \leftarrow true$
    end
    Find $l_j \xrightarrow{s_1} l_k \in F$
    Add $s_1$ to $M$
    if $l_1$ appears in $s_1$ then
        $s'_1 \leftarrow$ copy of $s_1$
        Mark any valid internal label of $s'_1$ as $*$-label
        Add $s'_1$ to $M$ /* case 1: *-label */
    else
        Find $l_1 \xrightarrow{s_2} l_j \in F$
        Add $s_2^{-1}s_2s_1$ to $M$ /* case 2: extends $s_2$ to reach $l_1$ */
    end
    if $swap$ then
        foreach $p \in M$ do
            Replace $p$ with $p^{-1}$
        end
    end
end
return $M$
```

a meta-walk $p'_i = s^{-1}sr$ in $T(D)$. By definition of FD, for each pair of entities e and f, $r(e, f, T(D))| = 1$ if a walk of r exists between e and f. Also, $|s^{-1}s(e, e, T(D))| = \sum_g |s^{-1}(e, g, T(D))| = \sum_g |s(g, e, T(D))|$. Thus, $|p'_i(e, f, T(D))| = |s^{-1}s(e, e, T(D))||r(e, f, T(D))| = (\sum_g |s(g, e, T(D))|)|r(e, f, T(D))| = \sum_g |s(g, e, T(D))| = \sum_g |[l_1, ..., l_j](e, g, D)| = |p_i(e, f, D)|$. Hence, the bijectivity of M holds with the desired properties. Therefore, the theorem holds. □

6 Empirical Evaluation

6.1 Experiment Settings

We use 5 datasets in our experiments. We use a subset of DBLP data with 1,227,602 nodes and 2,692,679 edges, which contains information about publications in computer science. We add information about the area for each conference in DBLP from Microsoft Academic Search. Figure 6a shows fragments of DBLP. We also use a subset of Microsoft Academic Search data with 44,044 nodes and 44,196 edges whose fragments are shown Figure 4a. We use Arxiv High Energy Physics paper citation graph from SNAP with 34,536 nodes and 42,158 edges whose fragments are shown in Figure 3b. We use a subset of IMDb data with 2,409,252 nodes and 7,525,281 edges whose fragments are shown in Figure 5a. We also use WSU course database from *cs.washington.edu/research/xmldatasets* with 1,124 nodes and 1,959 edges, which contains information about courses, instructors, and course offerings. Figure 7a shows fragments of this dataset. We implement our and other algorithms using MATLAB 8.5 on a Linux server with 64GB memory and two quad core processors.

6.2 Representation Independence

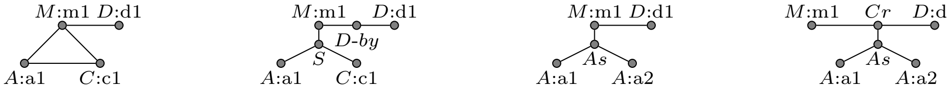
We use normalized Kendall's tau to compare ranked lists. The value of normalized Kendall's tau varies between 0 and 1 where 0 means the two lists are identical and 1 means one list is the reverse of the other. As users are interested in the highly ranked answers, we compare top 3, 5 and 10 answers.

Relationship Reorganization: Because it takes too long to run SimRank and RWR over full IMDb dataset, we use the largest subset of IMDb with 47,835 nodes and 130,916 edges over which we can run SimRank and RWR reasonably fast to evaluate their robustness. We set the restart probability of RWR and the damping

		IM2MV	IM2AS	IM2FB	DB2SI	WS2AL
Top 3	RWR	0.473	0.505	0.170	.482	.300
	SimRank	0.411	0.458	0.333	.481	.440
	PathSim	-	-	-	.641	.320
Top 5	RWR	0.444	0.459	0.158	.447	.259
	SimRank	0.365	0.392	0.337	.455	.387
	PathSim	-	-	-	.608	.310
Top 10	RWR	0.404	0.415	0.155	.412	.253
	SimRank	0.343	0.348	0.322	.410	.341
	PathSim	-	-	-	.590	.247

Table 1: Average ranking differences for all transformations.

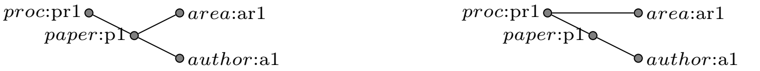
factor of SimRank to 0.8. We reorganize IMDb database to the structures of Freebase (FB), Movielicious (MVL) and a structure from *evc-cit.info/ cit041x/ assignment_ css.html* (ASM) whose fragments are shown in Figure 5b, Figure 5c and Figure 5d, respectively. We denote the transformations from IMDb to Freebase as IM2FB, from IMDb to Movielicious as IM2MV, and from IMDb to ASM as IM2AS. Since MVL and ASM structures do not have any *character*, we remove character nodes in IMDb when applying IM2MV and IM2AS transformations. For query workload, we randomly sample 50 movies in IMDb database based on their degrees. Table 1 shows the average ranking differences for top 3, 5, and 10 answers returned by RWR and SimRank



(a) IMDB (b) FB (c) MVL (d) ASGN

Figure 5: Fragments of movies databases where A, M, C, D, S, As, Cr and $D\text{-}by$ denotes *actor*, *movie*, *character*, *director*, *starring*, *actors*, *credit* and *directed-by*.

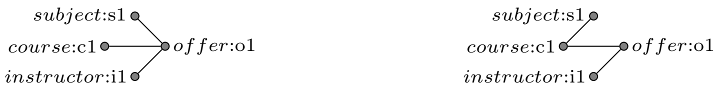
over IM2MV, IM2AS, and IM2FB transformations. Because R-PathSim delivers the same rankings over these transformations, we have omitted the results for R-PathSim. Because each entity label and its consecutive entity labels in every meta-walk over FB, MVL, and ASM data are different, all walks used in the computation of PathSim are informative, thus, PathSim is robust over these transformations according to Theorem 4.5. Hence, we omit the results of PathSim from the table. According to Table 1, the rankings produced by RWR and SimRank varies considerably over relationship-reorganizing transformations. As we have shown in Section 4, PathSim is not robust under certain relationship reorganizing transformations. We use the SNAP dataset and reorganize it to the structure of DBLP-citation as depicted in Figure 3a. For query workload, we randomly sample 50 papers from SNAP based on their degrees. We use [*paper*, *paper*, *paper*] meta-walk on SNAP and [*paper*, *citation*, *paper*, *citation*, *paper*] on DBLP-citation for PathSim and R-PathSim. The average ranking differences for top 3, 5 and 10 answers of PathSim are 0.564, 0.522 and 0.495, respectively. Hence, the output of PathSim varies significantly over some relationship-reorganizing transformations.



(a) DBLP (b) SIGMOD Rec. structure

Figure 6: Fragments of bibliographic databases.

Entity Rearrangement: We use DBLP and WSU course databases to evaluate the robustness of RWR, SimRank, PathSim, and R-PathSim over entity-rearranging transformations. Because SimRank and RWR take too long to finish on full DBLP dataset, we perform the following experiments using a subset of DBLP. with 24,396 nodes and 98,731 edges. The FDs in DBLP database are *paper* → *proc*, *paper* → *area*, and *proc* $\xrightarrow{[proc,paper,area]}$ *area*. We transform this database to a database that follows the structure of SIGMOD Record from *sigmod.org/publications*, where the information about each collection of papers is directly connected to the node that represents the collection. Figure 6b shows fragments of this database. The FDs in this database are *paper* → *proc*, *proc* → *area*, and *paper* $\xrightarrow{[paper,proc,area]}$ *area*. We call this transformation DB2SI. We randomly sample 100 proceedings based on their degrees in DBLP dataset as our query workload. The FDs in WSU database are *offer* → *course*, *offer* → *subject*, and *course* $\xrightarrow{[course,offer,subject]}$ *subject*. Figure 7 depicts the transformation of our WSU Course dataset to the structure of the Alchemy UW-CSE database from *alchemy.cs.washington.edu/data/uw-cse*. We call this transformation W2AL. The FDs in Alchemy UW-CSE database are *offer* → *course*, *offer* $\xrightarrow{[offer,course,subject]}$ *subject*, and *course* → *subject*. We randomly sample 100 courses from WSU based on their degrees as our query workload. Table 1 shows the average ranking differences



(a) WSU (b) Alchemy UW-CSE

Figure 7: Schemas for the course database.

for top 3, 5 and 10 answers from RWR, SimRank and PathSim under DB2SI and WS2AL. We use meta-walks [*proc*, *area*, *proc*] and [*proc*, *paper*, *area*, *paper*, *proc*] over DBLP and SIGMOD Record, respectively, for PathSim and R-PathSim. We use meta-walks [*course*, *offer*, *subject*, *offer*, *course*] and [*course*, *subject*, *course*] over WSU and Alchemy UW-CSE, respectively. Because R-PathSim returns the same answers over both transformations, we do not report its results. According to Table 1, the outputs of all algorithms, except R-PathSim, are significantly different over entity-rearranging transformations.

6.3 Efficiency and Effectiveness

Efficiency: We evaluate the efficiency of R-PathSim and PathSim over full IMDb and DBLP data. We transform IMDb to Movielicious (MVL) structure that contains both informative walks and non-informative walks to evaluate the impact of detecting informative walks in R-PathSim. This results in a database of 1,272,253 nodes and 2,886,494 edges. As we have explained in Section 6.2, DBLP dataset satisfy some FDs. Thus, we use it to measure the influence of using meta-walks with *-labels in the running time of R-PathSim. To explore the impact of both detecting informative walks and using meta-walks with *-labels on the efficiency of R-PathSim, we add a node without value, called *authors*, that groups authors of the same paper in DBLP dataset. This modification introduces non-informative walks to the database. We call the resulting database DBLP+, which contains 1,905,092 nodes and 3,370,169 edges. We precompute and store commuting matrices for meta-walks of size, i.e., number of labels, up to 3 to be used in query processing as done in PathSim [25]. MVL, DBLP, and DBLP+ have 16, 16, and 22 meta-walks with sizes less or equal to 3, respectively. It takes 49, 153, and 156 seconds for R-PathSim to precompute and store the commuting matrices of these meta-walk for MVL, DBLP, and DBLP+, respectively, which which are reasonable for a pre-processing step. We have executed PathSim over the same datasets and get almost equal running times as the ones of R-PathSim.

We randomly select 100 movies from MVL and 100 proceedings based on their degrees from DBLP and DBLP+ and use them as our query workloads. Because R-PathSim computes score over only informative meta-walks, we would like to measure the time used for the extra steps of detecting and ignoring non-informative walks. Because MVL and DBLP+ contains nodes without value, there exist non-informative walks in these two databases. Thus, we compare the query processing time of R-PathSim and PathSim over MVL and DBLP+. We first find a set of all *maximal* meta-walks of for given size over each database. Then we run R-PathSim and PathSim using these meta-walks, and compute the average time per query per meta-walk. Table 2 shows the average query processing of R-PathSim and PathSim per query per meta-walk given that commuting matrices up to size 3 are materialized. Overall, there is about 4% increase in running time of R-PathSim over PathSim due to an extra steps. Hence, the time spent on detecting and ignoring non-informative walks is almost negligible.

Next, we analyze and compare the efficiency of aggregated R-PathSim and aggregated PathSim. We use Algorithm 2 to constructs a subset of maximal meta-walks which R-PathSim computes aggregated score over. Since there is no algorithm presented in [25, 24] about finding a subset of meta-walks to be computed over, we find a subset of meta-walks up to a given size. Then we measure the running time of aggregated PathSim over these meta-walks.

Table 2 shows the query processing time of R-PathSim and PathSim per query, respectively, given that commuting matrices up to size 3 are materialized. The results indicate that the additional steps in R-PathSim to detect and ignore non-informative walks do not significantly increase its running time compared with that of PathSim.

Table 4 and Table 3 show the average query processing time of R-PathSim and PathSim per query per meta-walk given that commuting matrices up to size 3 are materialized. The reported processing time of R-PathSim also includes the time that Algorithm 2 constructs a subset of meta-walks which R-PathSim computes aggregated score over. The set of entity labels L for the input of Algorithm 2 is the set of all entity labels in the database. The results indicate that the running time of R-PathSim is reasonable for the design-time task when using $R < 3$ and assume that Algorithm 2 is run in the preprocessing steps.

Effectiveness: We evaluate the effectiveness of R-PathSim over the Microsoft Academic Search (MAS) dataset. For query workload, we randomly sample 100 conferences based on their degrees from the dataset. To provide the ground truth, given a conference q we manually group all other conferences in three categories: *similar*, which contains all conferences that have the same domain as q; *quite-similar*, which includes the conferences in the domains that are closely related to the domain of q; and *least-similar* that contain conferences

	Size	PathSim	R-PathSim
MVL	5	0.029	0.033
	7	0.021	0.023
DBLP+	5	0.026	0.010
	7	0.760	0.774

Table 2: Average query time in second between PathSim and R-PathSim per meta-walk of a given size.

	Size	Num	Time$_F$	Time$_Q$
MVL	5	3	0.118	0.099
	7	4	0.079	0.083
	9	8	0.119	6.082
DBLP	5	5	0.075	0.127
	7	7	0.076	5.329
	9	16	0.083	12.160
DBLP+	5	3	0.090	0.029
	7	6	0.077	4.644
	9	10	0.086	7.601

Table 3: Average query time (Time$_Q$) in second of aggregated PathSim. Time$_F$ denotes the time in finding a set of meta-walks of size up to the given size, and Num denotes the number of meta-walks found.

in the domains that are not strongly related to the domain of q. For example, *Data Mining* and *Databases* domains are strongly related, but *Databases* and *Computer Vision* are not. We use Normalized DCG (nDCG) to compare the effectiveness of R-PathSim and PathSim because it supports multiple levels of relevance for returned answers [21, 25]. The value of nDCG ranges between 0 and 1 where higher values show more effective ranking. We report the values of nDCG for top 5 (nDCG@5) and top 10 (nDCG@10) answers. In the first experiment, we use meta-walk [*conf*, *paper*, *citation*, *paper*, *citation*, *paper*, *conf*] to find similar conferences based on their papers' citations. Since R-PathSim considers only informative walks of this meta-walk, it will return different results than PathSim. The average nDCG@5 (nDCG@10) for R-PathSim and PathSim are .264 (.315) and .261 (.313) respectively. Although the value of nDCG for R-PathSim is higher than PathSim, the difference is not statistically significant according to the paired t-test at significant level of 0.05. In the second experiment, we evaluate the effectiveness of using meta-walks with $*$-labels. We compute the similarities of conferences based on the keywords in their domains. PathSim uses meta-walk [*conf*, *paper*, *domain*, *keyword*, *domain*, *paper*, *conf*] and R-PathSim uses meta-walk [*conf*, $\overline{paper}$, *domain*, *keyword*, *domain*, $\overline{paper}$, *conf*]. The average nDCG@5 (nDCG@10) for R-PathSim and PathSim are 1.0 (1.0) and 0.969 (0.901), respectively. R-PathSim significantly outperforms PathSim. Entities of type *paper* should not play a role in computing the similarity of conferences based on the keywords of their domains. Nevertheless, PathSim considers papers in determining these similarities. Hence, it deems conferences with more papers more similar, while they may not have that many common keywords. R-PathSim avoids this problem by treating *paper* as $*$-label. For example, the top 5 answers of R-PathSim for query *SIGKDD* are *ICDM*, *IDEAL*, *PAKDD*, *PJW* and *PKDD*. But, the top 5 answers of PathSim for the same query are *ICOMP*, *IC-AI*, *ICAIL*, *ICALP* and *ICANN*.

Next, we measure the effectiveness of aggragated R-PathSim over a set of meta-walks found by Algorithm 2. We generate a set of meta-walks over MAS using Algorithm 2 by giving a set of all entities in the dataset as an input and setting parameter R to 1 and 2. We compute the aggregated R-PathSim score over these mata-walks using the same query workload. The average nDCG@5 (nDCG@10) for R-PathSim using R equals to 1, 2 and 3 are 1.0 (1.0), 0.976 (0.932) and 0.936 (0.844), respectively. To analyze our effectiveness results, we also computed aggregated PathSim over a set of all meta-walks of size up to 5, 7 and 9 in the MAS database using the same query workload. The average nDCG@5 (nDCG@10) for PathSim over a subset of all meta-walks of size up to 5, 7 and 9 are 0.969 (0.901), 0.943 (0.852) and 0.933 (0.820), respectively. The results of R-PathSim using R equals to 1 are significantly better than the results of PathSim computed over meta-walks of size up to 5 and 7. The results of R-PathSim using R equals to 2 are significantly better than the results of PathSim

	R	Size	Num	Time$_F$	Time$_Q$
DBLP	1	5	4	0.273	0.145
	2	9	8	0.078	1.829
	3	13	12	1.214	60.975
DBLP+	1	7	4	4.103	0.858
	2	15	8	0.516	2.037
	3	23	12	1.594	58.811

Table 4: Average query time (Time$_Q$) in second of aggregated R-PathSim over DBLP and DBLP+. Time$_F$ denotes the time running Algorithm 2 using all entities labels as the input and parameter R. Size and Num denote the maximum size and total number of maximal meta-walks found by the algorithm.

computed over meta-walks of size up 7. There is no significant difference between any other results between PathSim and R-PathSim.

7 Conclusion

We postulated that a similarity search algorithm should return essentially the same answers for the same query over different representations of a database. We introduced two families of frequently occurring representational shifts over graph databases called relationship reorganizing and entity rearranging transformations. We showed that current well-known similarity search algorithms are not representation independence and propose new algorithms that are representation independent under these transformations.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley, 1994.

[2] I. Antonellis, H. Garcia-Molina, and C. Chang. Simrank++: Query Rewriting through Link Analysis of the Click Graph. In *VLDB*, 2008.

[3] M. Arenas and L. Libkin. A Normal Form for XML Documents. *TODS*, 29(1):195–232, 2004.

[4] P. Barcelo, J. Perez, and J. Reutter. Schema Mappings and Data Exchange for Graph Databases. In *ICDT*, pages 189–200, 2013.

[5] A. Borodin, G. Roberts, J. S. Rosenthal, and P. Tsaparas. Link Analysis Ranking: Algorithms, Theory, and Experiments. *ACM Trans. Inter. Tech.*, 5(1):231–297, 2005.

[6] E. Codd. Does Your DBMS Run By the Rules? *ComputerWorld*, 1985.

[7] W. Fan and P. Bohannon. Information Preserving XML Schema Embedding. *TODS*, 33(1):4:1–4:44, 2008.

[8] F. Geerts, H. Mannila, and E. Terzi. Relational Link-based Ranking. In *VLDB*, 2004.

[9] B. Ghadiri Bashardoost, C. Christodoulakis, S. Hassas Yeganeh, R. J. Miller, K. Lyons, and O. Hassanzadeh. Vizcurator: A visual tool for curating open data. In *WWW*, pages 195–198, 2015.

[10] G. Ghoshal and A. Barbasi. Ranking Stability and Super-stable Nodes in Complex Networks. *Nature Communications*, 2(394), 2011.

[11] C. Gutierrez, C. Hurtado, A. Mendelzon, and J. Perez. Foundations of Semantic Web Databases. *JCSS*, 77(3):520–541, 2010.

[12] J. Hayes and C. Gutierrez. Bipartite Graphs as Intermediate Model for RDF. In *ISWC*, pages 47–61, 2004.

[13] J. Heer, J. Hellerstein, and S. Kandel. Predictive interaction for data transformation. In *CIDR*, 2015.

[14] A. Hogana, M. Arenas, A. Mallea, and A. Polleres. Everything you always wanted to know about blank nodes. *Web Semantics*, pages 42–69, 2014.

[15] G. Jeh and J. Widom. SimRank: A Measure of Structural-context Similarity. In *KDD*, pages 538–543, 2002.

[16] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *CHI*, 2011.

[17] A. Khan, N. Li, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood Based Fast Graph Search in Large Networks. In *SIGMOD*, 2011.

[18] Y. Koren, S. North, and C. Volinsky. Measuring and Extracting Proximity in Networks. In *KDD*, 2006.

[19] Y. Koren, S. North, and C. Volinsky. Context-Aware Object Connection Discovery in Large Graphs. In *ICDE*, 2009.

[20] D. Liben-Nowelly and J. Kleinberg. The Link Prediction Problem for Social Networks. In *CIKM*, pages 556–559, 2003.

[21] C. Manning, P. Raghavan, and H. Schutze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[22] A. Ng, A. Zheng, and M. Jordan. Stable Algorithms for Link Analysis. In *SIGIR*, pages 556–559, 2001.

[23] P. Sarkar, A. Moore, and A. Prakash. Fast incremental proximity search in large graphs. In *ICML*, 2008.

[24] C. Shi, X. Kong, Y. Huang, S. Y. Philip, and B. Wu. Hetesim: A general framework for relevance measure in heterogeneous networks. *TKDE*.

[25] Y. Sun, J. Han, X. Yan, S. P. Yu, and T. Wu. PathSim: MetaPath-Based Top-K Similarity Search in Heterogeneous Information Networks. In *VLDB*, 2011.

[26] A. Termehchy, M. Winslett, Y. Chodpathumwan, and A. Gibbons. Design Independent Query Interfaces. *TKDE*, pages 1819–1832, 2012.

[27] Y. Tian and J. Patel. Tale: A Tool for Approximate Large Graph Matching. In *ICDE*, 2008.

[28] H. Tong and C. Faloutsos. Center-Piece Subgraphs: Problem Definition and Fast Solutions. In *KDD*, pages 404–413, 2006.

[29] H. Tong, C. Faloutsos, and J. Pan. Fast Random Walk with Restart and its Applications. In *ICDM*, pages 613–622, 2006.

[30] H. Tong, B. Gallagher, C. Faloutsos, and T. Eliassi-Rad. Fast Best-effort Pattern Matching in Large Attributed Graphs. In *KDD*, pages 737–746, 2007.

[31] H. Tong, H. Qu, and H. Jamjoom. Measuring Proximity on Graphs with Side Information. In *ICDM*, 2008.

[32] U. von Luxburg, A. Radl, and M. Hein. Hitting and Commute Times in Large Random Neighborhood Graphs. *JMLR*, 2014.

[33] S. H. Yeganeh, O. Hassanzadeh, and R. J. Miller. Linking Semistructured Data on the Web. In *WebDB*, 2011.

[34] C. Yu and H. V. Jagadish. Efficient Discovery of XML Data Redundancy. In *VLDB*, pages 103–114, 2006.

[35] P. Zhao, J. Han, and Y. Sun. P-Rank: A Comprehensive Structural Similarity Measure over Information Networks. In *CIKM*, pages 553–562, 2009.